Package ‘VGAM’

February 29, 2024

Version 1.1-10

Date 2024-02-29

Title Vector Generalized Linear and Additive Models

Author Thomas Yee [aut, cre] (<https://orcid.org/0000-0002-9970-3907>),
Cleve Moler [ctb] (author of several LINPACK routines)

Maintainer Thomas Yee <t.yee@auckland.ac.nz>

Depends R (>= 4.0.0), methods, stats, stats4, splines

Suggests VGAMextra, MASS, mgcv

Enhances VGAMdata

Description An implementation of about 6 major classes of statistical regression models. The central algorithm is Fisher scoring and iterative reweighted least squares. At the heart of this package are the vector generalized linear and additive model (VGLM/VGAM) classes. VGLMs can be loosely thought of as multivariate GLMs. VGAMs are data-driven VGLMs that use smoothing. The book ``Vector Generalized Linear and Additive Models: With an Implementation in R'' (Yee, 2015) <DOI:10.1007/978-1-4939-2818-7> gives details of the statistical framework and the package. Currently only fixed-effects models are implemented. Many (100+) models and distributions are estimated by maximum likelihood estimation (MLE) or penalized MLE. The other classes are RR-VGLMs (reduced-rank VGLMs), quadratic RR-VGLMs, doubly constrained RR-VGLMs, quadratic RR-VGLMs, reduced-rank VGAMs, RCIMs (row-column interaction models)---these classes perform constrained and unconstrained quadratic ordination (CQO/UQO) models in ecology, as well as constrained additive ordination (CAO). Hauck-Donner effect detection is implemented. Note that these functions are subject to change; see the NEWS and ChangeLog files for latest changes.

URL https://www.stat.auckland.ac.nz/~yee/VGAM/

NeedsCompilation yes

BuildVignettes yes

LazyLoad yes

LazyData yes

Repository CRAN

Date/Publication 2024-02-29 06:20:02 UTC

R topics documented:

VGAM-package 13
A1A2A3 17
AA.Aa.aa 18
AB.Ab.aB.ab 20
ABO 21
acat 22
add1.vglm 24
AICvlm 25
alaplace 27
alaplaceUC 32
alogitlink 33
altered 36
amlbinomial 37
amlexponential 39
amlnormal 41
amlpoisson 44
anova.vglm 46
AR1 48
AR1EIM 52
asinlink 55
auuc 57
aux.posbernoulli.t 58
backPain 59
beggs 60
bell 61
Benford 62
Benini 64
benini1 65
Betabinom 67
betabinomial 70
betabinomialff 73
betaff 76
Betageom 78
betageometric 79
betaII 81
Betanorm 82
betaprime 84
betaR 85

Biamhcop . . . 87
biamhcop . . . 88
Biclaytoncop . . . 90
biclaytoncop . . . 91
BICvlm . . . 93
Bifgmcop . . . 94
bifgmcop . . . 95
bifgmexp . . . 96
bifrankcop . . . 98
bigumbelIexp . . . 99
bilogis . . . 101
bilogistic . . . 102
Binom2.or . . . 104
binom2.or . . . 106
Binom2.rho . . . 109
binom2.rho . . . 111
binomialff . . . 113
Binorm . . . 116
binormal . . . 117
binormalcop . . . 119
Binormcop . . . 121
Biplackett . . . 122
biplackettcop . . . 123
biplot-methods . . . 125
Bisa . . . 125
bisa . . . 127
Bistudentt . . . 129
bistudentt . . . 130
bmi.nz . . . 131
borel.tanner . . . 132
Bort . . . 134
Brat . . . 135
brat . . . 136
bratt . . . 138
budworm . . . 140
calibrate . . . 141
calibrate-methods . . . 143
calibrate.qrrvglm . . . 143
calibrate.qrrvglm.control . . . 147
calibrate.rrvglm . . . 148
calibrate.rrvglm.control . . . 150
cao . . . 151
cao.control . . . 155
Card . . . 158
cardioid . . . 160
cauchitlink . . . 162
cauchy . . . 164
cdf.lmscreg . . . 166

cens.gumbel . . . 167
cens.normal . . . 169
cens.poisson . . . 171
cfibrosis . . . 173
cgo . . . 174
chest.nz . . . 175
chinese.nz . . . 176
chisq . . . 177
clo . . . 178
clogloglink . . . 180
CM.equid . . . 182
coalminers . . . 183
Coef . . . 184
Coef.qrrvglm . . . 185
Coef.qrrvglm-class . . . 187
Coef.rrvglm . . . 189
Coef.rrvglm-class . . . 190
Coef.vlm . . . 191
coefvgam . . . 192
coefvlm . . . 193
CommonVGAMffArguments . . . 194
concoef . . . 203
concoef-methods . . . 204
confintvglm . . . 205
constraints . . . 207
cops . . . 209
corbet . . . 210
cqo . . . 211
crashes . . . 218
cratio . . . 220
cumulative . . . 222
Dagum . . . 226
dagum . . . 228
dAR1 . . . 229
deermice . . . 231
deplot.lmscreg . . . 232
depvar . . . 234
dextlogF . . . 235
df.residual . . . 236
dgaitdplot . . . 237
dhuber . . . 240
Diffzeta . . . 242
diffzeta . . . 243
dirichlet . . . 244
dirmul.old . . . 246
dirmultinomial . . . 248
dlogF . . . 250
double.cens.normal . . . 251

double.expbinomial 253
ducklings 256
eCDF 256
enzyme 257
erf 258
erlang 259
Expectiles-Exponential 261
Expectiles-Normal 262
Expectiles-sc.t2 264
Expectiles-Uniform 265
expexpff 268
expexpff1 270
expgeom 272
expgeometric 273
expint 274
explink 276
explog 277
explogff 278
exponential 280
exppois 281
exppoisson 283
Extbetabinom 284
extbetabinomial 286
extlogF1 288
familyname 291
Felix 292
felix 293
fff 294
fill1 296
finney44 299
fisherzlink 300
Fisk 301
fisk 302
fittedvlm 304
fix.crossing 306
flourbeetle 307
Foldnorm 308
foldnormal 309
formulavlm 311
Frank 312
Frechet 314
frechet 315
freund61 317
Gaitdbinom 319
Gaitdlog 322
gaitdlog 324
Gaitdnbinom 327
gaitdnbinomial 329

Gaitdpois . . . 333
gaitdpoisson . . . 337
Gaitdzeta . . . 343
gaitdzeta . . . 346
gamma1 . . . 348
gamma2 . . . 350
gammaff.mm . . . 352
gammahyperbola . . . 354
gammaR . . . 355
garma . . . 357
GenbetaII . . . 359
genbetaII . . . 360
gengamma.stacy . . . 362
gengammaUC . . . 364
Genpois0 . . . 366
Genpois1 . . . 368
genpoisson0 . . . 369
genpoisson1 . . . 372
genpoisson2 . . . 373
genray . . . 375
genrayleigh . . . 376
Gensh . . . 378
gensh . . . 379
geometric . . . 381
get.smart . . . 382
get.smart.prediction . . . 383
gev . . . 384
gevUC . . . 387
gew . . . 389
goffset . . . 390
Gompertz . . . 392
gompertz . . . 393
gpd . . . 395
gpdUC . . . 398
grain.us . . . 400
grc . . . 401
gumbel . . . 406
Gumbel-II . . . 409
gumbelII . . . 410
gumbelUC . . . 412
guplot . . . 414
has.interceptvlm . . . 415
hatvalues . . . 416
hdeff . . . 418
hdeffsev . . . 422
hormone . . . 424
hspider . . . 426
huber2 . . . 427

Huggins89.t1 429
hunua 431
Hurea 433
hurea 434
hyperg 435
hypersecant 437
Hzeta 439
hzeta 440
iam 441
identitylink 443
Influence 444
inv.binomial 445
Inv.gaussian 446
inv.gaussianff 448
Inv.lomax 449
inv.lomax 451
Inv.paralogistic 452
inv.paralogistic 454
is.buggy 455
is.crossing 457
is.parallel 458
is.smart 459
is.zero 460
kendall.tau 461
KLD 462
Kumar 463
kumar 465
lakeO 466
lambertW 467
laplace 468
laplaceUC 470
latvar 472
leipnik 473
lerch 474
leukemia 476
levy 477
lgamma1 478
lgammaUC 480
Lindley 481
lindley 483
linkfun 484
Links 485
Lino 488
lino 490
lirat 492
lms.bcg 493
lms.bcn 495
lms.yjn 498

Log 501
log1mexp 502
logclink 503
logF 504
logff 506
logistic 508
logitlink 509
logitoffsetlink 512
loglaplace 513
loglapUC 517
logLik.vlm 519
loglinb2 520
loglinb3 522
loglink 523
logloglink 525
lognormal 526
logofflink 527
Lomax 529
lomax 530
lpossums 532
lqnorm 533
lrt.stat 535
lrtest 536
lvplot 538
lvplot.qrrvglm 539
lvplot.rrvglm 544
machinists 546
Makeham 548
makeham 549
margeff 551
marital.nz 553
Max 554
Maxwell 555
maxwell 557
mccullagh89 558
meangaitd 559
melbmaxtemp 561
meplot 562
micmen 564
mills.ratio 566
mix2exp 567
mix2normal 569
mix2poisson 571
MNSs 573
model.framevlm 574
model.matrixqrrvglm 576
model.matrixvlm 577
moffset 579

multilogitlink 581
multinomial 582
Nakagami 587
nakagami 588
nbcanlink 590
negbinomial 592
negbinomial.size 599
normal.vcm 601
nparam.vlm 604
olympics 605
Opt 607
ordpoisson 608
ordsup 610
oxtemp 612
Paralogistic 613
paralogistic 614
Pareto 616
paretoff 617
ParetoIV 619
paretoIV 621
Perks 624
perks 625
perspqrrvglm 627
pgamma.deriv 629
pgamma.deriv.unscaled 631
plotdeplot.lmscreg 632
plotdgaitd.vglm 634
plotqrrvglm 635
plotqtplot.lmscreg 636
plotrcim0 638
plotvgam 640
plotvgam.control 643
plotvglm 644
pneumo 646
poisson.points 647
poissonff 648
PoissonPoints 650
Polono 651
posbernoulli.b 653
posbernoulli.t 656
posbernoulli.tb 659
posbernUC 662
posbinomial 664
Posgeom 666
posnegbinomial 667
Posnorm 670
posnormal 672
pospoisson 674

powerlink . . . 675
prats . . . 676
predictqrrvglm . . . 678
predictvglm . . . 679
prentice74 . . . 681
prinia . . . 683
probitlink . . . 684
profilevglm . . . 686
propodds . . . 687
prplot . . . 688
put.smart . . . 690
qrrvglm.control . . . 691
qtplot.gumbel . . . 696
qtplot.lmscreg . . . 698
Qvar . . . 699
qvar . . . 703
R2latvar . . . 704
Rank . . . 705
Rayleigh . . . 706
rayleigh . . . 707
Rcim . . . 709
rcqo . . . 711
rdiric . . . 715
rec.exp1 . . . 716
rec.normal . . . 717
reciprocallink . . . 719
residualsvglm . . . 720
rhobitlink . . . 722
Rice . . . 724
riceff . . . 725
rigff . . . 727
rlplot.gevff . . . 728
rootogram4 . . . 730
round2 . . . 732
rrar . . . 733
rrvglm . . . 735
rrvglm-class . . . 738
rrvglm.control . . . 741
rrvglm.optim.control . . . 745
ruge . . . 746
s . . . 747
sc.studentt2 . . . 749
score.stat . . . 750
seglines . . . 752
Select . . . 753
seq2binomial . . . 756
setup.smart . . . 757
Simplex . . . 759

simplex 760
simulate.vlm 761
Sinmad 763
sinmad 764
Skellam 766
skellam 767
skewnorm 769
skewnormal 770
Slash 772
slash 773
sloglink 775
sm.os 776
sm.ps 780
smart.expression 783
smart.mode.is 783
smartpred 784
specials 786
spikeplot 787
sqrtlink 789
sratio 791
step4 794
studentt 795
summary.drrvglm 797
summarypvgam 800
summaryvgam 801
summaryvglm 802
SURff 805
SurvS4 807
SurvS4-class 809
TIC 810
Tobit 811
tobit 813
Tol 817
Topple 819
topple 820
toxop 821
Triangle 822
triangle 824
trim.constraints 826
Trinorm 828
trinormal 829
trplot 831
trplot.qrrvglm 832
Trunc 835
Truncpareto 836
truncweibull 838
ucberk 840
uninormal 841

UtilitiesVGAM . . . 843
V1 . . . 844
V2 . . . 845
vcovvlm . . . 846
venice . . . 848
vgam . . . 850
vgam-class . . . 854
vgam.control . . . 857
vglm . . . 859
vglm-class . . . 865
vglm.control . . . 868
vglmff-class . . . 872
vonmises . . . 875
vplot.profile . . . 877
vsmooth.spline . . . 878
waitakere . . . 880
wald.stat . . . 882
waldff . . . 884
weibull.mean . . . 885
weibullR . . . 887
weightsvglm . . . 890
wine . . . 891
wrapup.smart . . . 892
yeo.johnson . . . 893
Yules . . . 894
yulesimon . . . 896
Zabinom . . . 897
zabinomial . . . 898
Zageom . . . 900
zageometric . . . 901
Zanegbin . . . 903
zanegbinomial . . . 905
Zapois . . . 908
zapoisson . . . 909
zero . . . 911
Zeta . . . 913
zeta . . . 914
zetaff . . . 916
Zibinom . . . 918
zibinomial . . . 919
Zigeom . . . 922
zigeometric . . . 923
Zinegbin . . . 925
zinegbinomial . . . 927
zipebcom . . . 930
Zipf . . . 933
zipf . . . 934
Zipfmb . . . 936

Zipois 937
zipoisson 939
Zoabeta 943
zoabetaR 945

Index **947**

`VGAM-package` *Vector Generalized Linear and Additive Models and Other Associated Models*

Description

VGAM provides functions for fitting vector generalized linear and additive models (VGLMs and VGAMs), and associated models (Reduced-rank VGLMs or RR-VGLMs, Doubly constrained RR-VGLMs (DRR-VGLMs), Quadratic RR-VGLMs, Reduced-rank VGAMs). This package fits many models and distributions by maximum likelihood estimation (MLE) or penalized MLE, under this statistical framework. Also fits constrained ordination models in ecology such as constrained quadratic ordination (CQO).

Details

This package centers on the *iteratively reweighted least squares* (IRLS) algorithm. Other key words include Fisher scoring, additive models, reduced-rank regression, penalized likelihood, and constrained ordination. The central modelling functions are `vglm`, `vgam`, `rrvglm`, `rcim`, `cqo`, `cao`. Function `vglm` operates very similarly to `glm` but is much more general, and many methods functions such as `coef` and `predict` are available. The package uses S4 (see `methods-package`).

Some notable companion packages: (1) **VGAMdata** mainly contains data sets useful for illustrating **VGAM**. Some of the big ones were initially from **VGAM**. Recently, some older **VGAM** family functions have been shifted into this package. (2) **VGAMextra** written by Victor Miranda has some additional **VGAM** family and link functions, with a bent towards time series models. (3) **svyVGAM** provides design-based inference, e.g., to survey sampling settings. This is because the `weights` argument of `vglm` can be assigned any positive values including survey weights.

Compared to other similar packages, such as **gamlss** and **mgcv**, **VGAM** has more models implemented (150+ of them) and they are not restricted to a location-scale-shape framework or (largely) the 1-parameter exponential family. The general statistical framework behind it all, once grasped, makes regression modelling unified. Some features of the package are: (i) many family functions handle multiple responses; (ii) reduced-rank regression is available by operating on latent variables (optimal linear combinations of the explanatory variables); (iii) basic automatic smoothing parameter selection is implemented for VGAMs (`sm.os` and `sm.ps` with a call to `magic`), although it has to be refined; (iv) *smart* prediction allows correct prediction of nested terms in the formula provided smart functions are used.

The GLM and GAM classes are special cases of VGLMs and VGAMs. The VGLM/VGAM framework is intended to be very general so that it encompasses as many distributions and models as possible. VGLMs are limited only by the assumption that the regression coefficients enter through a set of linear predictors. The VGLM class is very large and encompasses a wide range of multivariate response types and models, e.g., it includes univariate and multivariate distributions, categorical data analysis, extreme values, correlated binary data, quantile and expectile regression, time series problems. Potentially, it can handle generalized estimating equations, survival analysis, bioassay data and nonlinear least-squares problems.

Crudely, VGAMs are to VGLMs what GAMs are to GLMs. Two types of VGAMs are implemented: 1st-generation VGAMs with `s` use vector backfitting, while 2nd-generation VGAMs with `sm.os` and `sm.ps` use O-splines and P-splines so have a direct solution (hence avoids backfitting) and have automatic smoothing parameter selection. The former is older and is based on Yee and Wild (1996). The latter is more modern (Yee, Somchit and Wild, 2024) but it requires a reasonably large number of observations to work well because it is based on optimizing over a predictive criterion rather than using a Bayesian approach.

An important feature of the framework is that of *constraint matrices*. They apportion the regression coefficients according to each explanatory variable. For example, since each parameter has a link function applied to it to turn it into a linear or additive predictor, does a covariate have an equal effect on each parameter? Or no effect? Arguments such as `zero`, `parallel` and `exchangeable`, are merely easy ways to have them constructed internally. Users may input them explicitly using the `constraint` argument, and `CM.symm0` etc. can make this easier.

Another important feature is implemented by `xij`. It allows different linear/additive predictors to have a different values of the same explanatory variable, e.g., `multinomial` for the conditional logit model and the like.

VGLMs with dimension reduction form the class of RR-VGLMs. This is achieved by reduced rank regression. Here, a subset of the constraint matrices are estimated rather than being known and prespecified. Optimal linear combinations of the explanatory variables are taken (creating latent variables) which are used for fitting a VGLM. Thus the regression can be thought of as being in two stages. The class of DRR-VGLMs provides further structure to RR-VGLMs by allowing constraint matrices to be specified for each column of $\mathbf{A}$ and row of $\mathbf{C}$. Thus the reduced rank regression can be fitted with greater control.

This package is the first to check for the *Hauck-Donner effect* (HDE) in regression models; see `hdeff`. This is an aberration of the Wald statistics when the parameter estimates are too close to the boundary of the parameter space. When present the p-value of a regression coefficient is biased upwards so that a highly significant variable might be deemed nonsignificant. Thus the HDE can create havoc for variable selection!

Somewhat related to the previous paragraph, hypothesis testing using the likelihood ratio test, Rao's score test (Lagrange multiplier test) and (modified) Wald's test are all available; see `summaryvglm`. For all regression coefficients of a model, taken one at a time, all three methods require further IRLS iterations to obtain new values of the other regression coefficients after one of the coefficients has had its value set (usually to 0). Hence the computation load is overall significant.

For a complete list of this package, use `library(help = "VGAM")`. New **VGAM** family functions are continually being written and added to the package.

Warning

This package is undergoing continual development and improvement, therefore users should treat many things as subject to change. This includes the family function names, argument names, many of the internals, moving some functions to **VGAMdata**, the use of link functions, and slot names. For example, many link functions were renamed in 2019 so that they all end in `"link"`, e.g., `loglink()` instead of `loge()`. Some future pain can be avoided by using good programming techniques, e.g., using extractor functions such as `coef()`, `weights()`, `vcov()`, `predict()`. Although changes are now less frequent, please expect changes in all aspects of the package. See the NEWS file for a list of changes from version to version.

Author(s)

Thomas W. Yee, <t.yee@auckland.ac.nz>, with contributions from Victor Miranda and several graduate students over the years, especially Xiangjie (Albert) Xue and Chanatda Somchit.

Maintainer: Thomas Yee <t.yee@auckland.ac.nz>.

References

Yee, T. W. (2015). *Vector Generalized Linear and Additive Models: With an Implementation in R*. New York, USA: *Springer*.

Yee, T. W. and Hastie, T. J. (2003). Reduced-rank vector generalized linear models. *Statistical Modelling*, **3**, 15–41.

Yee, T. W. and Stephenson, A. G. (2007). Vector generalized linear and additive extreme value models. *Extremes*, **10**, 1–19.

Yee, T. W. and Wild, C. J. (1996). Vector generalized additive models. *Journal of the Royal Statistical Society, Series B, Methodological*, **58**, 481–493.

Yee, T. W. (2004). A new technique for maximum-likelihood canonical Gaussian ordination. *Ecological Monographs*, **74**, 685–701.

Yee, T. W. (2006). Constrained additive ordination. *Ecology*, **87**, 203–213.

Yee, T. W. (2008). The VGAM Package. *R News*, **8**, 28–39.

Yee, T. W. (2010). The **VGAM** package for categorical data analysis. *Journal of Statistical Software*, **32**, 1–34. doi:10.18637/jss.v032.i10.

Yee, T. W. (2014). Reduced-rank vector generalized linear models with two linear predictors. *Computational Statistics and Data Analysis*, **71**, 889–902.

Yee, T. W. and Ma, C. (2024). Generally altered, inflated, truncated and deflated regression. *Statistical Science*, **39** (in press).

Yee, T. W. (2022). On the Hauck-Donner effect in Wald tests: Detection, tipping points and parameter space characterization, *Journal of the American Statistical Association*, **117**, 1763–1774. doi:10.1080/01621459.2021.1886936.

Yee, T. W. and Somchit, C. and Wild, C. J. (2024). Penalized vector generalized additive models. Manuscript in preparation.

The website for the **VGAM** package and book is https://www.stat.auckland.ac.nz/~yee/. There are some resources there, especially as relating to my book and new features added to **VGAM**.

Some useful background reference for the package include:

Chambers, J. and Hastie, T. (1991). *Statistical Models in S*. Wadsworth & Brooks/Cole.

Green, P. J. and Silverman, B. W. (1994). *Nonparametric Regression and Generalized Linear Models: A Roughness Penalty Approach*. Chapman and Hall.

Hastie, T. J. and Tibshirani, R. J. (1990). *Generalized Additive Models*. Chapman and Hall.

See Also

vglm, vgam, rrvglm, rcim, cqo, TypicalVGAMfamilyFunction, CommonVGAMffArguments, Links, hdeff, glm, lm, https://CRAN.R-project.org/package=VGAM.

Examples

```
# Example 1; proportional odds model
pneumo <- transform(pneumo, let = log(exposure.time))
(fit1 <- vglm(cbind(normal, mild, severe) ~ let, propodds, data = pneumo))
depvar(fit1)  # Better than using fit1@y; dependent variable (response)
weights(fit1, type = "prior")  # Number of observations
coef(fit1, matrix = TRUE)      # p.179, in McCullagh and Nelder (1989)
constraints(fit1)              # Constraint matrices
summary(fit1)  # HDE could affect these results
summary(fit1, lrt0 = TRUE, score0 = TRUE, wald0 = TRUE)  # No HDE
hdeff(fit1)  # Check for any Hauck-Donner effect

# Example 2; zero-inflated Poisson model
zdata <- data.frame(x2 = runif(nn <- 2000))
zdata <- transform(zdata, pstr0  = logitlink(-0.5 + 1*x2, inverse = TRUE),
                          lambda = loglink(  0.5 + 2*x2, inverse = TRUE))
zdata <- transform(zdata, y = rzipois(nn, lambda, pstr0 = pstr0))
with(zdata, table(y))
fit2 <- vglm(y ~ x2, zipoisson, data = zdata, trace = TRUE)
coef(fit2, matrix = TRUE)  # These should agree with the above values


# Example 3; fit a two species GAM simultaneously
fit3 <- vgam(cbind(agaaus, kniexc) ~ s(altitude, df = c(2, 3)),
             binomialff(multiple.responses = TRUE), data = hunua)
coef(fit3, matrix = TRUE)   # Not really interpretable
## Not run:  plot(fit3, se = TRUE, overlay = TRUE, lcol = 3:4, scol = 3:4)

ooo <- with(hunua, order(altitude))
with(hunua,  matplot(altitude[ooo], fitted(fit3)[ooo, ], type = "l",
     lwd = 2, col = 3:4,
     xlab = "Altitude (m)", ylab = "Probability of presence", las = 1,
     main = "Two plant species' response curves", ylim = c(0, 0.8)))
with(hunua, rug(altitude))
## End(Not run)


# Example 4; LMS quantile regression
fit4 <- vgam(BMI ~ s(age, df = c(4, 2)), lms.bcn(zero = 1),
             data = bmi.nz, trace = TRUE)
head(predict(fit4))
head(fitted(fit4))
head(bmi.nz)  # Person 1 is near the lower quartile among people his age
head(cdf(fit4))

## Not run:  par(mfrow = c(1,1), bty = "l", mar = c(5,4,4,3)+0.1, xpd=TRUE)
qtplot(fit4, percentiles = c(5,50,90,99), main = "Quantiles", las = 1,
```

```
      xlim = c(15, 90), ylab = "BMI", lwd=2, lcol=4)  # Quantile plot

ygrid <- seq(15, 43, len = 100)  # BMI ranges
par(mfrow = c(1, 1), lwd = 2)  # Density plot
aa <- deplot(fit4, x0 = 20, y = ygrid, xlab = "BMI", col = "black",
    main = "Density functions at Age=20 (black), 42 (red) and 55 (blue)")
aa
aa <- deplot(fit4, x0 = 42, y = ygrid, add = TRUE, llty = 2, col = "red")
aa <- deplot(fit4, x0 = 55, y = ygrid, add = TRUE, llty = 4, col = "blue",
            Attach = TRUE)
aa@post$deplot  # Contains density function values

## End(Not run)


# Example 5; GEV distribution for extremes
(fit5 <- vglm(maxtemp ~ 1, gevff, data = oxtemp, trace = TRUE))
head(fitted(fit5))
coef(fit5, matrix = TRUE)
Coef(fit5)
vcov(fit5)
vcov(fit5, untransform = TRUE)
sqrt(diag(vcov(fit5)))  # Approximate standard errors
## Not run:  rlplot(fit5)
```

A1A2A3 *The A1A2A3 Blood Group System*

Description

Estimates the three independent parameters of the the A1A2A3 blood group system.

Usage

```
A1A2A3(link = "logitlink", inbreeding = FALSE, ip1 = NULL, ip2 = NULL, iF = NULL)
```

Arguments

`link`	Link function applied to `p1`, `p2` and `f`. See `Links` for more choices.
`inbreeding`	Logical. Is there inbreeding?
`ip1, ip2, iF`	Optional initial value for `p1`, `p2` and `f`.

Details

The parameters `p1` and `p2` are probabilities, so that `p3=1-p1-p2` is the third probability. The parameter `f` is the third independent parameter if `inbreeding = TRUE`. If `inbreeding = FALSE` then $f = 0$ and Hardy-Weinberg Equilibrium (HWE) is assumed.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm and vgam.

Note

The input can be a 6-column matrix of counts, with columns corresponding to A1A1, A1A2, A1A3, A2A2, A2A3, A3A3 (in order). Alternatively, the input can be a 6-column matrix of proportions (so each row adds to 1) and the weights argument is used to specify the total number of counts for each row.

Author(s)

T. W. Yee

References

Lange, K. (2002). *Mathematical and Statistical Methods for Genetic Analysis*, 2nd ed. New York: Springer-Verlag.

See Also

AA.Aa.aa, AB.Ab.aB.ab, ABO, MNSs.

Examples

```
ymat <- cbind(108, 196, 429, 143, 513, 559)
fit <- vglm(ymat ~ 1, A1A2A3(link = probitlink), trace = TRUE, crit = "coef")
fit <- vglm(ymat ~ 1, A1A2A3(link = logitlink, ip1 = 0.3, ip2 = 0.3, iF = 0.02),
            trace = TRUE, crit = "coef")
Coef(fit)  # Estimated p1 and p2
rbind(ymat, sum(ymat) * fitted(fit))
sqrt(diag(vcov(fit)))
```

AA.Aa.aa *The AA-Aa-aa Blood Group System*

Description

Estimates the parameter of the AA-Aa-aa blood group system, with or without Hardy Weinberg equilibrium.

Usage

```
AA.Aa.aa(linkp = "logitlink", linkf = "logitlink", inbreeding = FALSE,
         ipA = NULL, ifp = NULL, zero = NULL)
```

Arguments

linkp, linkf	Link functions applied to pA and f. See Links for more choices.
ipA, ifp	Optional initial values for pA and f.
inbreeding	Logical. Is there inbreeding?
zero	See CommonVGAMffArguments for information.

Details

This one or two parameter model involves a probability called pA. The probability of getting a count in the first column of the input (an AA) is pA*pA. When inbreeding = TRUE, an additional parameter f is used. If inbreeding = FALSE then $f = 0$ and Hardy-Weinberg Equilibrium (HWE) is assumed. The EIM is used if inbreeding = FALSE.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm and vgam.

Warning

Setting inbreeding = FALSE makes estimation difficult with non-intercept-only models. Currently, this code seems to work with intercept-only models.

Note

The input can be a 3-column matrix of counts, where the columns are AA, Ab and aa (in order). Alternatively, the input can be a 3-column matrix of proportions (so each row adds to 1) and the weights argument is used to specify the total number of counts for each row.

Author(s)

T. W. Yee

References

Weir, B. S. (1996). *Genetic Data Analysis II: Methods for Discrete Population Genetic Data*, Sunderland, MA: Sinauer Associates, Inc.

See Also

AB.Ab.aB.ab, ABO, A1A2A3, MNSs.

Examples

```
y <- cbind(53, 95, 38)
fit1 <- vglm(y ~ 1, AA.Aa.aa, trace = TRUE)
fit2 <- vglm(y ~ 1, AA.Aa.aa(inbreeding = TRUE), trace = TRUE)
rbind(y, sum(y) * fitted(fit1))
Coef(fit1)  # Estimated pA
```

```
Coef(fit2)  # Estimated pA and f
summary(fit1)
```

AB.Ab.aB.ab *The AB-Ab-aB-ab Blood Group System*

Description

Estimates the parameter of the AB-Ab-aB-ab blood group system.

Usage

```
AB.Ab.aB.ab(link = "logitlink", init.p = NULL)
```

Arguments

`link`	Link function applied to `p`. See `Links` for more choices.
`init.p`	Optional initial value for `p`.

Details

This one parameter model involves a probability called `p`.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Note

The input can be a 4-column matrix of counts, where the columns are AB, Ab, aB and ab (in order). Alternatively, the input can be a 4-column matrix of proportions (so each row adds to 1) and the `weights` argument is used to specify the total number of counts for each row.

Author(s)

T. W. Yee

References

Lange, K. (2002). *Mathematical and Statistical Methods for Genetic Analysis*, 2nd ed. New York: Springer-Verlag.

See Also

`AA.Aa.aa`, `ABO`, `A1A2A3`, `MNSs`.

Examples

```
ymat <- cbind(AB=1997, Ab=906, aB=904, ab=32)  # Data from Fisher (1925)
fit <- vglm(ymat ~ 1, AB.Ab.aB.ab(link = "identitylink"), trace = TRUE)
fit <- vglm(ymat ~ 1, AB.Ab.aB.ab, trace = TRUE)
rbind(ymat, sum(ymat)*fitted(fit))
Coef(fit)  # Estimated p
p <- sqrt(4*(fitted(fit)[, 4]))
p*p
summary(fit)
```

ABO *The ABO Blood Group System*

Description

Estimates the two independent parameters of the the ABO blood group system.

Usage

```
ABO(link.pA = "logitlink", link.pB = "logitlink", ipA = NULL, ipB = NULL,
    ipO = NULL, zero = NULL)
```

Arguments

`link.pA, link.pB`	Link functions applied to `pA` and `pB`. See `Links` for more choices.
`ipA, ipB, ipO`	Optional initial value for `pA` and `pB` and `pO`. A `NULL` value means values are computed internally.
`zero`	Details at `CommonVGAMffArguments`.

Details

The parameters `pA` and `pB` are probabilities, so that `pO=1-pA-pB` is the third probability. The probabilities `pA` and `pB` correspond to A and B respectively, so that `pO` is the probability for O. It is easier to make use of initial values for `pO` than for `pB`. In documentation elsewhere I sometimes use `pA=p`, `pB=q`, `pO=r`.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Note

The input can be a 4-column matrix of counts, where the columns are A, B, AB, O (in order). Alternatively, the input can be a 4-column matrix of proportions (so each row adds to 1) and the `weights` argument is used to specify the total number of counts for each row.

Author(s)

T. W. Yee

References

Lange, K. (2002). *Mathematical and Statistical Methods for Genetic Analysis*, 2nd ed. New York: Springer-Verlag.

See Also

`AA.Aa.aa`, `AB.Ab.aB.ab`, `A1A2A3`, `MNSs`.

Examples

```
ymat <- cbind(A = 725, B = 258, AB = 72, O = 1073)  # Order matters, not the name
fit <- vglm(ymat ~ 1, ABO(link.pA = "identitylink",
                          link.pB = "identitylink"), trace = TRUE,
            crit = "coef")
coef(fit, matrix = TRUE)
Coef(fit)  # Estimated pA and pB
rbind(ymat, sum(ymat) * fitted(fit))
sqrt(diag(vcov(fit)))
```

`acat` *Ordinal Regression with Adjacent Categories Probabilities*

Description

Fits an adjacent categories regression model to an ordered (preferably) factor response.

Usage

```
acat(link = "loglink", parallel = FALSE, reverse = FALSE,
     zero = NULL, ynames = FALSE, Thresh = NULL, Trev = reverse,
     Tref = if (Trev) "M" else 1, whitespace = FALSE)
```

Arguments

`link`	Link function applied to the ratios of the adjacent categories probabilities. See `Links` for more choices.
`parallel`	A logical, or formula specifying which terms have equal/unequal coefficients.
`reverse`	Logical. By default, the linear/additive predictors used are $\eta_j = \log(P[Y = j+1]/P[Y = j])$ for $j = 1, \ldots, M$. If `reverse` is `TRUE` then $\eta_j = \log(P[Y = j]/P[Y = j+1])$ will be used.
`ynames`	See `multinomial` for information.

`zero` An integer-valued vector specifying which linear/additive predictors are modelled as intercepts only. The values must be from the set $\{1,2,\ldots,M\}$. See `CommonVGAMffArguments` for more information.

`Thresh, Trev, Tref` See `cumulative` for information. These arguments apply to ordinal categorical regression models.

`whitespace` See `CommonVGAMffArguments` for information.

Details

In this help file the response Y is assumed to be a factor with ordered values $1, 2, \ldots, M+1$, so that M is the number of linear/additive predictors η_j. By default, the log link is used because the ratio of two probabilities is positive.

Internally, `deriv3` is called to perform symbolic differentiation and consequently this family function will struggle if M becomes too large. If this occurs, try combining levels so that M is effectively reduced. One idea is to aggregate levels with the fewest observations in them first.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, `rrvglm` and `vgam`.

Warning

No check is made to verify that the response is ordinal if the response is a matrix; see `ordered`.

Note

The response should be either a matrix of counts (with row sums that are all positive), or an ordered factor. In both cases, the `y` slot returned by `vglm`/`vgam`/`rrvglm` is the matrix of counts.

For a nominal (unordered) factor response, the multinomial logit model (`multinomial`) is more appropriate.

Here is an example of the usage of the `parallel` argument. If there are covariates `x1`, `x2` and `x3`, then `parallel = TRUE ~ x1 + x2 -1` and `parallel = FALSE ~ x3` are equivalent. This would constrain the regression coefficients for `x1` and `x2` to be equal; those of the intercepts and `x3` would be different.

Author(s)

Thomas W. Yee

References

Agresti, A. (2013). *Categorical Data Analysis*, 3rd ed. Hoboken, NJ, USA: Wiley.
Tutz, G. (2012). *Regression for Categorical Data*, Cambridge: Cambridge University Press.
Yee, T. W. (2010). The **VGAM** package for categorical data analysis. *Journal of Statistical Software*, **32**, 1–34. doi:10.18637/jss.v032.i10.

See Also

cumulative, cratio, sratio, multinomial, CM.equid, CommonVGAMffArguments, margeff, pneumo, budworm, deriv3.

Examples

```
pneumo <- transform(pneumo, let = log(exposure.time))
(fit <- vglm(cbind(normal, mild, severe) ~ let, acat, pneumo))
coef(fit, matrix = TRUE)
constraints(fit)
model.matrix(fit)
```

---

add1.vglm *Add or Drop All Possible Single Terms to/from a Model*

Description

Compute all the single terms in the scope argument that can be added to or dropped from the model, fit those models and compute a table of the changes in fit.

Usage

```
## S3 method for class 'vglm'
add1(object, scope, test = c("none", "LRT"), k = 2, ...)
## S3 method for class 'vglm'
drop1(object, scope, test = c("none", "LRT"), k = 2, ...)
```

Arguments

object	a fitted vglm model object.
scope, k	See drop1.glm.
test	Same as drop1.glm but with fewer choices.
...	further arguments passed to or from other methods.

Details

These functions are a direct adaptation of add1.glm and drop1.glm for vglm-class objects. For drop1 methods, a missing scope is taken to be all terms in the model. The hierarchy is respected when considering terms to be added or dropped: all main effects contained in a second-order interaction must remain, and so on. In a scope formula . means 'what is already there'.

Compared to add1.glm and drop1.glm these functions are simpler, e.g., there is no *Cp*, F and Rao (score) tests, x and scale arguments. Most models do not have a deviance, however twice the log-likelihood differences are used to test the significance of terms.

The default output table gives AIC, defined as minus twice log likelihood plus $2p$ where p is the rank of the model (the number of effective parameters). This is only defined up to an additive constant (like log-likelihoods).

Value

An object of class "anova" summarizing the differences in fit between the models.

Warning

In general, the same warnings in add1.glm and drop1.glm apply here. Furthermore, these functions have not been rigorously tested for all models, so treat the results cautiously and please report any bugs.

Care is needed to check that the constraint matrices of added terms are correct. Also, if object is of the form vglm(..., constraints = list(x1 = cm1, x2 = cm2)) then add1.vglm may fail because the constraints argument needs to have the constaint matrices for *all* terms.

Note

Most **VGAM** family functions do not compute a deviance, but instead the likelihood function is evaluated at the MLE. Hence a column name "Deviance" only appears for a few models; and almost always there is a column labelled "logLik".

See Also

step4vglm, vglm, extractAIC.vglm, trim.constraints, anova.vglm, backPain2, update.

Examples

```
data("backPain2", package = "VGAM")
summary(backPain2)
fit1 <- vglm(pain ~ x2 + x3 + x4, propodds, data = backPain2)
coef(fit1)
add1(fit1, scope = ~ x2 * x3 * x4, test = "LRT")
drop1(fit1, test = "LRT")
fit2 <- vglm(pain ~ x2 * x3 * x4, propodds, data = backPain2)
drop1(fit2)
```

AICvlm *Akaike's Information Criterion*

Description

Calculates the Akaike information criterion for a fitted model object for which a log-likelihood value has been obtained.

Usage

```
    AICvlm(object, ..., corrected = FALSE, k = 2)
   AICvgam(object, ..., k = 2)
 AICrrvglm(object, ..., k = 2)
AICqrrvglm(object, ..., k = 2)
 AICrrvgam(object, ..., k = 2)
```

Arguments

`object`	Some **VGAM** object, for example, having class `vglm-class`.
`...`	Other possible arguments fed into `logLik` in order to compute the log-likelihood.
`corrected`	Logical, perform the finite sample correction?
`k`	Numeric, the penalty per parameter to be used; the default is the classical AIC.

Details

The following formula is used for VGLMs: $-2\text{log-likelihood} + kn_{par}$, where n_{par} represents the number of parameters in the fitted model, and $k = 2$ for the usual AIC. One could assign $k = \log(n)$ (n the number of observations) for the so-called BIC or SBC (Schwarz's Bayesian criterion). This is the function `AICvlm()`.

This code relies on the log-likelihood being defined, and computed, for the object. When comparing fitted objects, the smaller the AIC, the better the fit. The log-likelihood and hence the AIC is only defined up to an additive constant.

Any estimated scale parameter (in GLM parlance) is used as one parameter.

For VGAMs and CAO the nonlinear effective degrees of freedom for each smoothed component is used. This formula is heuristic. These are the functions `AICvgam()` and `AICcao()`.

The finite sample correction is usually recommended when the sample size is small or when the number of parameters is large. When the sample size is large their difference tends to be negligible. The correction is described in Hurvich and Tsai (1989), and is based on a (univariate) linear model with normally distributed errors.

Value

Returns a numeric value with the corresponding AIC (or BIC, or . . . , depending on `k`).

Warning

This code has not been double-checked. The general applicability of `AIC` for the VGLM/VGAM classes has not been developed fully. In particular, `AIC` should not be run on some **VGAM** family functions because of violation of certain regularity conditions, etc.

Note

AIC has not been defined for QRR-VGLMs, yet.

Using AIC to compare `posbinomial` models with, e.g., `posbernoulli.tb` models, requires `posbinomial(omit.constant = TRUE)`. See `posbinomial` for an example. A warning is given if it suspects a wrong `omit.constant` value was used.

Where defined, `AICc(...)` is the same as `AIC(..., corrected = TRUE)`.

Author(s)

T. W. Yee.

References

Hurvich, C. M. and Tsai, C.-L. (1989). Regression and time series model selection in small samples, *Biometrika*, **76**, 297–307.

See Also

VGLMs are described in `vglm-class`; VGAMs are described in `vgam-class`; RR-VGLMs are described in `rrvglm-class`; `AIC`, `BICvlm`, `TICvlm`, `drop1.vglm`, `extractAIC.vglm`.

Examples

```
pneumo <- transform(pneumo, let = log(exposure.time))
(fit1 <- vglm(cbind(normal, mild, severe) ~ let,
              cumulative(parallel = TRUE, reverse = TRUE), data = pneumo))
coef(fit1, matrix = TRUE)
AIC(fit1)
AICc(fit1)  # Quick way
AIC(fit1, corrected = TRUE)  # Slow way
(fit2 <- vglm(cbind(normal, mild, severe) ~ let,
              cumulative(parallel = FALSE, reverse = TRUE), data = pneumo))
coef(fit2, matrix = TRUE)
AIC(fit2)
AICc(fit2)
AIC(fit2, corrected = TRUE)
```

`alaplace` *Asymmetric Laplace Distribution Family Functions*

Description

Maximum likelihood estimation of the 1, 2 and 3-parameter asymmetric Laplace distributions (ALDs). The 2-parameter ALD may, with trepidation and lots of skill, sometimes be used as an approximation of quantile regression.

Usage

```
alaplace1(tau = NULL, llocation = "identitylink",
          ilocation = NULL, kappa = sqrt(tau/(1 - tau)), Scale.arg = 1,
          ishrinkage = 0.95, parallel.locat = TRUE  ~ 0, digt = 4,
          idf.mu = 3, zero = NULL, imethod = 1)

alaplace2(tau = NULL,  llocation = "identitylink", lscale = "loglink",
          ilocation = NULL, iscale = NULL, kappa = sqrt(tau/(1 - tau)),
          ishrinkage = 0.95,
          parallel.locat =  TRUE ~ 0,
          parallel.scale = FALSE ~ 0,
          digt = 4, idf.mu = 3, imethod = 1, zero = "scale")
```

```
alaplace3(llocation = "identitylink", lscale = "loglink",
          lkappa = "loglink", ilocation = NULL, iscale = NULL,
          ikappa = 1, imethod = 1, zero = c("scale", "kappa"))
```

Arguments

`tau, kappa` Numeric vectors with $0 < \tau < 1$ and $\kappa > 0$. Most users will only specify `tau` since the estimated location parameter corresponds to the τth regression quantile, which is easier to understand. See below for details.

`llocation, lscale, lkappa` Character. Parameter link functions for location parameter ξ, scale parameter σ, asymmetry parameter κ. See Links for more choices. For example, the argument `llocation` can help handle count data by restricting the quantiles to be positive (use `llocation = "loglink"`). However, `llocation` is best left alone since the theory only works properly with the identity link.

`ilocation, iscale, ikappa` Optional initial values. If given, it must be numeric and values are recycled to the appropriate length. The default is to choose the value internally.

`parallel.locat, parallel.scale` See the `parallel` argument of CommonVGAMffArguments. These arguments apply to the location and scale parameters. It generally only makes sense for the scale parameters to be equal, hence set `parallel.scale = TRUE`. Note that assigning `parallel.locat` the value `TRUE` circumvents the seriously embarrassing quantile crossing problem because all constraint matrices except for the intercept correspond to a parallelism assumption.

`imethod` Initialization method. Either the value 1, 2, 3 or 4.

`idf.mu` Degrees of freedom for the cubic smoothing spline fit applied to get an initial estimate of the location parameter. See vsmooth.spline. Used only when `imethod = 3`.

`ishrinkage` How much shrinkage is used when initializing ξ. The value must be between 0 and 1 inclusive, and a value of 0 means the individual response values are used, and a value of 1 means the median or mean is used. This argument is used only when `imethod = 4`. See CommonVGAMffArguments for more information.

`Scale.arg` The value of the scale parameter σ. This argument may be used to compute quantiles at different τ values from an existing fitted `alaplace2()` model (practical only if it has a single value). If the model has `parallel.locat = TRUE` then only the intercept need be estimated; use an offset. See below for an example.

`digt` Passed into Round as the `digits` argument for the `tau` values; used cosmetically for labelling.

`zero` See CommonVGAMffArguments for more information. Where possible, the default is to model all the σ and κ as an intercept-only term. See CommonVGAMffArguments for more information.

Details

These **VGAM** family functions implement one variant of asymmetric Laplace distributions (ALDs) suitable for quantile regression. Kotz et al. (2001) call it *the* ALD. Its density function is

$$f(y; \xi, \sigma, \kappa) = \frac{\sqrt{2}}{\sigma} \, \frac{\kappa}{1 + \kappa^2} \, \exp\left(-\frac{\sqrt{2}}{\sigma \, \kappa} |y - \xi|\right)$$

for $y \le \xi$, and

$$f(y; \xi, \sigma, \kappa) = \frac{\sqrt{2}}{\sigma} \, \frac{\kappa}{1 + \kappa^2} \, \exp\left(-\frac{\sqrt{2} \, \kappa}{\sigma} |y - \xi|\right)$$

for $y > \xi$. Here, the ranges are for all real y and ξ, positive σ and positive κ. The special case $\kappa = 1$ corresponds to the (symmetric) Laplace distribution of Kotz et al. (2001). The mean is $\xi + \sigma(1/\kappa - \kappa)/\sqrt{2}$ and the variance is $\sigma^2(1 + \kappa^4)/(2\kappa^2)$. The enumeration of the linear/additive predictors used for `alaplace2()` is the first location parameter followed by the first scale parameter, then the second location parameter followed by the second scale parameter, etc. For `alaplace3()`, only a vector response is handled and the last (third) linear/additive predictor is for the asymmetry parameter.

It is known that the maximum likelihood estimate of the location parameter ξ corresponds to the regression quantile estimate of the classical quantile regression approach of Koenker and Bassett (1978). An important property of the ALD is that $P(Y \le \xi) = \tau$ where $\tau = \kappa^2/(1 + \kappa^2)$ so that $\kappa = \sqrt{\tau/(1 - \tau)}$. Thus `alaplace2()` might be used as an alternative to `rq` in the **quantreg** package, although scoring is really an unsuitable algorithm for estimation here.

Both `alaplace1()` and `alaplace2()` can handle multiple responses, and the number of linear/additive predictors is dictated by the length of `tau` or `kappa`. The functions `alaplace1()` and `alaplace2()` can also handle multiple responses (i.e., a matrix response) but only with a *single-valued* `tau` or `kappa`.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

In the `extra` slot of the fitted object are some list components which are useful, e.g., the sample proportion of values which are less than the fitted quantile curves.

Warning

These functions are *experimental* and especially subject to change or withdrawal. The usual MLE regularity conditions do *not* hold for this distribution so that misleading inferences may result, e.g., in the `summary` and `vcov` of the object. The 1-parameter ALD can be approximated by `extlogF1` which has continuous derivatives and is recommended over `alaplace1`.

Care is needed with `tau` values which are too small, e.g., for count data with `llocation = "loglink"` and if the sample proportion of zeros is greater than `tau`.

Note

These **VGAM** family functions use Fisher scoring. Convergence may be slow and half-stepping is usual (although one can use `trace = TRUE` to see which is the best model and then use `maxit` to choose that model) due to the regularity conditions not holding. Often the iterations slowly crawl towards the solution so monitoring the convergence (set `trace = TRUE`) is highly recommended. Instead, `extlogF1` is recommended.

For large data sets it is a very good idea to keep the length of `tau`/`kappa` low to avoid large memory requirements. Then for `parallel.locat = FALSE` one can repeatedly fit a model with `alaplace1()` with one τ at a time; and for `parallel.locat = TRUE` one can refit a model with `alaplace1()` with one τ at a time but using offsets and an intercept-only model.

A second method for solving the noncrossing quantile problem is illustrated below in Example 3. This is called the *accumulative quantile method* (AQM) and details are in Yee (2015). It does not make the strong parallelism assumption.

The functions `alaplace2()` and `laplace` differ slightly in terms of the parameterizations.

Author(s)

Thomas W. Yee

References

Koenker, R. and Bassett, G. (1978). Regression quantiles. *Econometrica*, **46**, 33–50.

Kotz, S., Kozubowski, T. J. and Podgorski, K. (2001). *The Laplace distribution and generalizations: a revisit with applications to communications, economics, engineering, and finance*, Boston: Birkhauser.

See Also

`ralap`, `laplace`, `extlogF1`, `CommonVGAMffArguments`, `lms.bcn`, `amlnormal`, `sc.studentt2`, `simulate.vlm`.

Examples

```
## Not run:
# Example 1: quantile regression with smoothing splines
set.seed(123); adata <- data.frame(x2 = sort(runif(n <- 500)))
mymu <- function(x) exp(-2 + 6*sin(2*x-0.2) / (x+0.5)^2)
adata <- transform(adata, y = rpois(n, lambda = mymu(x2)))
mytau <- c(0.25, 0.75); mydof <- 4

fit <- vgam(y ~ s(x2, df = mydof), data=adata, trace=TRUE, maxit = 900,
            alaplace2(tau = mytau, llocat = "loglink",
                      parallel.locat = FALSE))
fitp <- vgam(y ~ s(x2, df = mydof), data = adata, trace=TRUE, maxit=900,
     alaplace2(tau = mytau, llocat = "loglink", parallel.locat = TRUE))

par(las = 1); mylwd <- 1.5
with(adata, plot(x2, jitter(y, factor = 0.5), col = "orange",
                 main = "Example 1; green: parallel.locat = TRUE",
                 ylab = "y", pch = "o", cex = 0.75))
with(adata, matlines(x2, fitted(fit ), col = "blue",
                     lty = "solid", lwd = mylwd))
with(adata, matlines(x2, fitted(fitp), col = "green",
                     lty = "solid", lwd = mylwd))
```

```
finexgrid <- seq(0, 1, len = 1001)
for (ii in 1:length(mytau))
  lines(finexgrid, qpois(p = mytau[ii], lambda = mymu(finexgrid)),
        col = "blue", lwd = mylwd)
fit@extra  # Contains useful information


# Example 2: regression quantile at a new tau value from an existing fit
# Nb. regression splines are used here since it is easier.
fitp2 <- vglm(y ~ sm.bs(x2, df = mydof), data = adata, trace = TRUE,
              alaplace1(tau = mytau, llocation = "loglink",
                        parallel.locat = TRUE))

newtau <- 0.5  # Want to refit the model with this tau value
fitp3 <- vglm(y ~ 1 + offset(predict(fitp2)[, 1]),
              alaplace1(tau = newtau, llocation = "loglink"), adata)
with(adata, plot(x2, jitter(y, factor = 0.5), col = "orange",
               pch = "o", cex = 0.75, ylab = "y",
               main = "Example 2; parallel.locat = TRUE"))
with(adata, matlines(x2, fitted(fitp2), col = "blue",
                     lty = 1, lwd = mylwd))
with(adata, matlines(x2, fitted(fitp3), col = "black",
                     lty = 1, lwd = mylwd))


# Example 3: noncrossing regression quantiles using a trick: obtain
# successive solutions which are added to previous solutions; use a log
# link to ensure an increasing quantiles at any value of x.

mytau <- seq(0.2, 0.9, by = 0.1)
answer <- matrix(0, nrow(adata), length(mytau))  # Stores the quantiles
adata <- transform(adata, offsety = y*0)
usetau <- mytau
for (ii in 1:length(mytau)) {
# cat("\n\nii  = ", ii, "\n")
  adata <- transform(adata, usey = y-offsety)
  iloc <- ifelse(ii == 1, with(adata, median(y)), 1.0)  # Well-chosen!
  mydf <- ifelse(ii == 1, 5, 3)  # Maybe less smoothing will help
  fit3 <- vglm(usey ~ sm.ns(x2, df = mydf), data = adata, trace = TRUE,
            alaplace2(tau = usetau[ii], lloc = "loglink", iloc = iloc))
  answer[, ii] <- (if(ii == 1) 0 else answer[, ii-1]) + fitted(fit3)
  adata <- transform(adata, offsety = answer[, ii])
}

# Plot the results.
with(adata, plot(x2, y, col = "blue",
     main = paste("Noncrossing and nonparallel; tau  = ",
                paste(mytau, collapse = ", "))))
with(adata, matlines(x2, answer, col = "orange", lty = 1))

# Zoom in near the origin.
with(adata, plot(x2, y, col = "blue", xlim = c(0, 0.2), ylim = 0:1,
     main = paste("Noncrossing and nonparallel; tau  = ",
```

```
                paste(mytau, collapse = ", "))))
with(adata, matlines(x2, answer, col = "orange", lty = 1))

## End(Not run)
```

alaplaceUC *The Laplace Distribution*

Description

Density, distribution function, quantile function and random generation for the 3-parameter asymmetric Laplace distribution with location parameter `location`, scale parameter `scale`, and asymmetry parameter `kappa`.

Usage

```
dalap(x, location = 0, scale = 1, tau = 0.5, kappa = sqrt(tau/(1-tau)),
      log = FALSE)
palap(q, location = 0, scale = 1, tau = 0.5, kappa = sqrt(tau/(1-tau)),
      lower.tail = TRUE, log.p = FALSE)
qalap(p, location = 0, scale = 1, tau = 0.5, kappa = sqrt(tau/(1-tau)),
      lower.tail = TRUE, log.p = FALSE)
ralap(n, location = 0, scale = 1, tau = 0.5, kappa = sqrt(tau/(1-tau)))
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. If `length(n) > 1` then the length is taken to be the number required.
`location`	the location parameter ξ.
`scale`	the scale parameter σ. Must consist of positive values.
`tau`	the quantile parameter τ. Must consist of values in $(0, 1)$. This argument is used to specify `kappa` and is ignored if `kappa` is assigned.
`kappa`	the asymmetry parameter κ. Must consist of positive values.
`log`	if `TRUE`, probabilities `p` are given as `log(p)`.
`lower.tail, log.p`	Same meaning as in `pnorm` or `qnorm`.

Details

There are many variants of asymmetric Laplace distributions (ALDs) and this one is known as *the* ALD by Kotz et al. (2001). See `alaplace3`, the **VGAM** family function for estimating the three parameters by maximum likelihood estimation, for formulae and details. The ALD density may be approximated by `dextlogF`.

Value

`dalap` gives the density, `palap` gives the distribution function, `qalap` gives the quantile function, and `ralap` generates random deviates.

Author(s)

T. W. Yee and Kai Huang

References

Kotz, S., Kozubowski, T. J. and Podgorski, K. (2001). *The Laplace distribution and generalizations: a revisit with applications to communications, economics, engineering, and finance*, Boston: Birkhauser.

See Also

`alaplace3`, `dextlogF`, `extlogF1`.

Examples

```
x <- seq(-5, 5, by = 0.01)
loc <- 0; sigma <- 1.5; kappa <- 2
## Not run:  plot(x, dalap(x, loc, sigma, kappa = kappa), type = "l",
     main = "Blue is density, orange is the CDF",
     ylim = c(0, 1), sub = "Purple are 5, 10, ..., 95 percentiles",
     las = 1, ylab = "", cex.main = 0.5, col = "blue")
abline(h = 0, col = "blue", lty = 2)
lines(qalap(seq(0.05, 0.95, by = 0.05), loc, sigma, kappa = kappa),
      dalap(qalap(seq(0.05, 0.95, by = 0.05), loc, sigma, kappa = kappa),
            loc, sigma, kappa = kappa), col="purple", lty=3, type = "h")
lines(x, palap(x, loc, sigma, kappa = kappa), type = "l", col = "orange")
abline(h = 0, lty = 2)
## End(Not run)

pp <- seq(0.05, 0.95, by = 0.05)  # Test two functions
max(abs(palap(qalap(pp, loc, sigma, kappa = kappa),
              loc, sigma, kappa = kappa) - pp))  # Should be 0
```

`alogitlink` *Arcsine–Logit Link Mixtures*

Description

Computes some arcsine–logit mixture link transformations, including their inverse and the first few derivatives.

Usage

```
  alogitlink(theta, bvalue = NULL, taumix.logit = 1,
     tol = 1e-13, nmax = 99, inverse = FALSE, deriv = 0,
     short = TRUE, tag = FALSE, c10 = c(4, -pi))
lcalogitlink(theta, bvalue = NULL, pmix.logit = 0.01,
     tol = 1e-13, nmax = 99, inverse = FALSE, deriv = 0,
     short = TRUE, tag = FALSE, c10 = c(4, -pi))
```

Arguments

`theta`	Numeric or character. See below for further details.
`bvalue`	See `Links`.
`taumix.logit`	Numeric, of length 1. Mixing parameter assigned to `logitlink`. Then `1 - exp(-taumix.log * theta)` is used to weight `asinlink`. Thus a 0 value will result in `logitlink` and a very large numeric such as `1e4` should be roughly equivalent to `asinlink` over almost all of the parameter space.
`pmix.logit`	Numeric, of length 1. Mixing probability assigned to `logitlink`. Then `1 - pmix.logit` is used to weight `asinlink`. Thus a 0 value will result in `asinlink`. and 1 is equivalent to `logitlink`.
`tol, nmax`	Arguments fed into a function implementing a vectorized bisection method.
`inverse, deriv, short, tag`	Details at `Links`.
`c10`	See `asinlink` and `logitlink`.

Details

`lcalogitlink` is a *linear combination* (LC) of `asinlink` and `logitlink`.

Value

The following holds for the LC variant. For `deriv >= 0`, `(1 - pmix.logit) * asinlink(p, deriv = deriv) + pmix.logit * logitlink(p, deriv = deriv)` when `inverse = FALSE`, and if `inverse = TRUE` then a nonlinear equation is solved for the probability, given `eta`. For `deriv = 1`, then the function returns *d* `eta` / *d* `theta` as a function of `theta` if `inverse = FALSE`, else if `inverse = TRUE` then it returns the reciprocal.

Warning

The default values for `taumix.logit` and `pmix.logit` may change in the future. The name and order of the arguments may change too.

Author(s)

Thomas W. Yee

References

Hauck, J. W. W. and A. Donner (1977). Wald's test as applied to hypotheses in logit analysis. *Journal of the American Statistical Association*, **72**, 851–853.

See Also

asinlink, logitlink, Links, probitlink, clogloglink, cauchitlink, binomialff, sloglink, hdeff, https://www.cia.gov/index.html.

Examples

```
p <- seq(0.01, 0.99, length= 10)
alogitlink(p)
max(abs(alogitlink(alogitlink(p), inv = TRUE) - p))  # 0?

## Not run:
par(mfrow = c(2, 2), lwd = (mylwd <- 2))
y <- seq(-4, 4, length = 100)
p <- seq(0.01, 0.99, by = 0.01)

for (d in 0:1) {
  matplot(p, cbind(logitlink(p, deriv = d), probitlink(p, deriv = d)),
          type = "n", col = "blue", ylab = "transformation",
          las = 1, main = if (d == 0) "Some probability link functions"
          else "First derivative")
  lines(p,   logitlink(p, deriv = d), col = "green")
  lines(p,  probitlink(p, deriv = d), col = "blue")
  lines(p, clogloglink(p, deriv = d), col = "tan")
  lines(p,  alogitlink(p, deriv = d), col = "red3")
  if (d == 0) {
    abline(v = 0.5, h = 0, lty = "dashed")
    legend(0, 4.5, c("logitlink", "probitlink", "clogloglink",
           "alogitlink"), lwd = mylwd,
           col = c("green", "blue", "tan", "red3"))
  } else
    abline(v = 0.5, lwd = 0.5, col = "gray")
}

for (d in 0) {
  matplot(y, cbind( logitlink(y, deriv = d, inverse = TRUE),
                   probitlink(y, deriv = d, inverse = TRUE)),
          type  = "n", col = "blue", xlab = "transformation", ylab = "p",
          main = if (d == 0) "Some inverse probability link functions"
          else "First derivative", las=1)
  lines(y,   logitlink(y, deriv = d, inverse = TRUE), col = "green")
  lines(y,  probitlink(y, deriv = d, inverse = TRUE), col = "blue")
  lines(y, clogloglink(y, deriv = d, inverse = TRUE), col = "tan")
  lines(y,  alogitlink(y, deriv = d, inverse = TRUE), col = "red3")
  if (d == 0) {
      abline(h = 0.5, v = 0, lwd = 0.5, col = "gray")
      legend(-4, 1, c("logitlink", "probitlink", "clogloglink",
             "alogitlink"), lwd = mylwd,
```

```
                col = c("green", "blue", "tan", "red3"))
  }
}
par(lwd = 1)

## End(Not run)
```

---

altered *Altered, Inflated, Truncated and Deflated Values in GAITD Regression*

Description

Return the altered, inflated, truncated and deflated values in a GAITD regression object, else test whether the model is altered, inflated, truncated or deflated.

Usage

```
altered(object, ...)
inflated(object, ...)
truncated(object, ...)
is.altered(object, ...)
is.deflated(object, ...)
is.inflated(object, ...)
is.truncated(object, ...)
```

Arguments

`object`	an object of class `"vglm"`. Currently only a GAITD regression object returns valid results of these functions.
`...`	any additional arguments, to future-proof this function.

Details

Yee and Ma (2023) propose GAITD regression where values from four (or seven since there are parametric and nonparametric forms) disjoint sets are referred to as *special*. These extractor functions return one set each; they are the `alter`, `inflate`, `truncate`, `deflate` (and sometimes `max.support`) arguments from the family function.

Value

Returns one type of 'special' sets associated with GAITD regression. This is a vector, else a list for truncation. All three sets are returned by `specialsvglm`.

Warning

Some of these functions are subject to change. Only family functions beginning with `"gaitd"` will work with these functions, hence `zipoisson` fits will return `FALSE` or empty values.

References

Yee, T. W. and Ma, C. (2024). Generally altered, inflated, truncated and deflated regression. *Statistical Science*, **39** (in press).

See Also

vglm, vglm-class, specialsvglm, gaitdpoisson, gaitdlog, gaitdzeta, Gaitdpois.

Examples

```
abdata <- data.frame(y = 0:7, w = c(182, 41, 12, 2, 2, 0, 0, 1))
fit1 <- vglm(y ~ 1, gaitdpoisson(a.mix = 0),
             data = abdata, weight = w, subset = w > 0)
specials(fit1)  # All three sets
altered(fit1)  # Subject to change
inflated(fit1)  # Subject to change
truncated(fit1)  # Subject to change
is.altered(fit1)
is.inflated(fit1)
is.truncated(fit1)
```

`amlbinomial` *Binomial Logistic Regression by Asymmetric Maximum Likelihood Estimation*

Description

Binomial quantile regression estimated by maximizing an asymmetric likelihood function.

Usage

```
amlbinomial(w.aml = 1, parallel = FALSE, digw = 4, link = "logitlink")
```

Arguments

`w.aml`	Numeric, a vector of positive constants controlling the percentiles. The larger the value the larger the fitted percentile value (the proportion of points below the "w-regression plane"). The default value of unity results in the ordinary maximum likelihood (MLE) solution.
`parallel`	If `w.aml` has more than one value then this argument allows the quantile curves to differ by the same amount as a function of the covariates. Setting this to be `TRUE` should force the quantile curves to not cross (although they may not cross anyway). See `CommonVGAMffArguments` for more information.
`digw`	Passed into `Round` as the `digits` argument for the `w.aml` values; used cosmetically for labelling.
`link`	See `binomialff`.

Details

The general methodology behind this **VGAM** family function is given in Efron (1992) and full details can be obtained there. This model is essentially a logistic regression model (see `binomialff`) but the usual deviance is replaced by an asymmetric squared error loss function; it is multiplied by $w.aml$ for positive residuals. The solution is the set of regression coefficients that minimize the sum of these deviance-type values over the data set, weighted by the `weights` argument (so that it can contain frequencies). Newton-Raphson estimation is used here.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Warning

If `w.aml` has more than one value then the value returned by `deviance` is the sum of all the (weighted) deviances taken over all the `w.aml` values. See Equation (1.6) of Efron (1992).

Note

On fitting, the `extra` slot has list components `"w.aml"` and `"percentile"`. The latter is the percent of observations below the "w-regression plane", which is the fitted values. Also, the individual deviance values corresponding to each element of the argument `w.aml` is stored in the `extra` slot.

For `amlbinomial` objects, methods functions for the generic functions `qtplot` and `cdf` have not been written yet.

See `amlpoisson` about comments on the jargon, e.g., *expectiles* etc.

In this documentation the word *quantile* can often be interchangeably replaced by *expectile* (things are informal here).

Author(s)

Thomas W. Yee

References

Efron, B. (1992). Poisson overdispersion estimates based on the method of asymmetric maximum likelihood. *Journal of the American Statistical Association*, **87**, 98–107.

See Also

`amlpoisson`, `amlexponential`, `amlnormal`, `extlogF1`, `alaplace1`, `denorm`.

Examples

```
# Example: binomial data with lots of trials per observation
set.seed(1234)
sizevec <- rep(100, length = (nn <- 200))
mydat <- data.frame(x = sort(runif(nn)))
mydat <- transform(mydat,
```

```
                    prob = logitlink(-0 + 2.5*x + x^2, inverse = TRUE))
mydat <- transform(mydat, y = rbinom(nn, size = sizevec, prob = prob))
(fit <- vgam(cbind(y, sizevec - y) ~ s(x, df = 3),
             amlbinomial(w = c(0.01, 0.2, 1, 5, 60)),
             mydat, trace = TRUE))
fit@extra

## Not run:
par(mfrow = c(1,2))
# Quantile plot
with(mydat, plot(x, jitter(y), col = "blue", las = 1, main =
     paste(paste(round(fit@extra$percentile, digits = 1), collapse = ", "),
           "percentile-expectile curves")))
with(mydat, matlines(x, 100 * fitted(fit), lwd = 2, col = "blue", lty=1))

# Compare the fitted expectiles with the quantiles
with(mydat, plot(x, jitter(y), col = "blue", las = 1, main =
     paste(paste(round(fit@extra$percentile, digits = 1), collapse = ", "),
           "percentile curves are red")))
with(mydat, matlines(x, 100 * fitted(fit), lwd = 2, col = "blue", lty = 1))

for (ii in fit@extra$percentile)
    with(mydat, matlines(x, 100 *
         qbinom(p = ii/100, size = sizevec, prob = prob) / sizevec,
                  col = "red", lwd = 2, lty = 1))

## End(Not run)
```

`amlexponential` *Exponential Regression by Asymmetric Maximum Likelihood Estimation*

Description

Exponential expectile regression estimated by maximizing an asymmetric likelihood function.

Usage

```
amlexponential(w.aml = 1, parallel = FALSE, imethod = 1, digw = 4,
                link = "loglink")
```

Arguments

`w.aml` Numeric, a vector of positive constants controlling the expectiles. The larger the value the larger the fitted expectile value (the proportion of points below the "w-regression plane"). The default value of unity results in the ordinary maximum likelihood (MLE) solution.

`parallel` If `w.aml` has more than one value then this argument allows the quantile curves to differ by the same amount as a function of the covariates. Setting this to be `TRUE` should force the quantile curves to not cross (although they may not cross anyway). See `CommonVGAMffArguments` for more information.

`imethod` Integer, either 1 or 2 or 3. Initialization method. Choose another value if convergence fails.

`digw` Passed into `Round` as the `digits` argument for the `w.aml` values; used cosmetically for labelling.

`link` See `exponential` and the warning below.

Details

The general methodology behind this **VGAM** family function is given in Efron (1992) and full details can be obtained there.

This model is essentially an exponential regression model (see `exponential`) but the usual deviance is replaced by an asymmetric squared error loss function; it is multiplied by $w.aml$ for positive residuals. The solution is the set of regression coefficients that minimize the sum of these deviance-type values over the data set, weighted by the `weights` argument (so that it can contain frequencies). Newton-Raphson estimation is used here.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Warning

Note that the `link` argument of `exponential` and `amlexponential` are currently different: one is the rate parameter and the other is the mean (expectile) parameter.

If `w.aml` has more than one value then the value returned by `deviance` is the sum of all the (weighted) deviances taken over all the `w.aml` values. See Equation (1.6) of Efron (1992).

Note

On fitting, the `extra` slot has list components `"w.aml"` and `"percentile"`. The latter is the percent of observations below the "w-regression plane", which is the fitted values. Also, the individual deviance values corresponding to each element of the argument `w.aml` is stored in the `extra` slot.

For `amlexponential` objects, methods functions for the generic functions `qtplot` and `cdf` have not been written yet.

See `amlpoisson` about comments on the jargon, e.g., *expectiles* etc.

In this documentation the word *quantile* can often be interchangeably replaced by *expectile* (things are informal here).

Author(s)

Thomas W. Yee

References

Efron, B. (1992). Poisson overdispersion estimates based on the method of asymmetric maximum likelihood. *Journal of the American Statistical Association*, **87**, 98–107.

See Also

exponential, amlbinomial, amlpoisson, amlnormal, extlogF1, alaplace1, lms.bcg, deexp.

Examples

```
nn <- 2000
mydat <- data.frame(x = seq(0, 1, length = nn))
mydat <- transform(mydat,
                   mu = loglink(-0 + 1.5*x + 0.2*x^2, inverse = TRUE))
mydat <- transform(mydat, mu = loglink(0 - sin(8*x), inverse = TRUE))
mydat <- transform(mydat,  y = rexp(nn, rate = 1/mu))
(fit <- vgam(y ~ s(x, df=5), amlexponential(w=c(0.001, 0.1, 0.5, 5, 60)),
             mydat, trace = TRUE))
fit@extra

## Not run:  # These plots are against the sqrt scale (to increase clarity)
par(mfrow = c(1,2))
# Quantile plot
with(mydat, plot(x, sqrt(y), col = "blue", las = 1, main =
     paste(paste(round(fit@extra$percentile, digits = 1), collapse=", "),
           "percentile-expectile curves")))
with(mydat, matlines(x, sqrt(fitted(fit)), lwd = 2, col = "blue", lty=1))

# Compare the fitted expectiles with the quantiles
with(mydat, plot(x, sqrt(y), col = "blue", las = 1, main =
     paste(paste(round(fit@extra$percentile, digits = 1), collapse=", "),
           "percentile curves are orange")))
with(mydat, matlines(x, sqrt(fitted(fit)), lwd = 2, col = "blue", lty=1))

for (ii in fit@extra$percentile)
  with(mydat, matlines(x, sqrt(qexp(p = ii/100, rate = 1/mu)),
                       col = "orange"))
## End(Not run)
```

`amlnormal` *Asymmetric Least Squares Quantile Regression*

Description

Asymmetric least squares, a special case of maximizing an asymmetric likelihood function of a normal distribution. This allows for expectile/quantile regression using asymmetric least squares error loss.

Usage

```
amlnormal(w.aml = 1, parallel = FALSE, lexpectile = "identitylink",
          iexpectile = NULL, imethod = 1, digw = 4)
```

Arguments

`w.aml` Numeric, a vector of positive constants controlling the percentiles. The larger the value the larger the fitted percentile value (the proportion of points below the "w-regression plane"). The default value of unity results in the ordinary least squares (OLS) solution.

`parallel` If `w.aml` has more than one value then this argument allows the quantile curves to differ by the same amount as a function of the covariates. Setting this to be `TRUE` should force the quantile curves to not cross (although they may not cross anyway). See `CommonVGAMffArguments` for more information.

`lexpectile, iexpectile` See `CommonVGAMffArguments` for more information.

`imethod` Integer, either 1 or 2 or 3. Initialization method. Choose another value if convergence fails.

`digw` Passed into `Round` as the `digits` argument for the `w.aml` values; used cosmetically for labelling.

Details

This is an implementation of Efron (1991) and full details can be obtained there. Equation numbers below refer to that article. The model is essentially a linear model (see `lm`), however, the asymmetric squared error loss function for a residual r is r^2 if $r \leq 0$ and wr^2 if $r > 0$. The solution is the set of regression coefficients that minimize the sum of these over the data set, weighted by the `weights` argument (so that it can contain frequencies). Newton-Raphson estimation is used here.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Note

On fitting, the `extra` slot has list components `"w.aml"` and `"percentile"`. The latter is the percent of observations below the "w-regression plane", which is the fitted values.

One difficulty is finding the `w.aml` value giving a specified percentile. One solution is to fit the model within a root finding function such as `uniroot`; see the example below.

For `amlnormal` objects, methods functions for the generic functions `qtplot` and `cdf` have not been written yet.

See the note in `amlpoisson` on the jargon, including *expectiles* and *regression quantiles*.

The `deviance` slot computes the total asymmetric squared error loss (2.5). If `w.aml` has more than one value then the value returned by the slot is the sum taken over all the `w.aml` values.

This **VGAM** family function could well be renamed `amlnormal()` instead, given the other function names `amlpoisson`, `amlbinomial`, etc.

In this documentation the word *quantile* can often be interchangeably replaced by *expectile* (things are informal here).

Author(s)

Thomas W. Yee

References

Efron, B. (1991). Regression percentiles using asymmetric squared error loss. *Statistica Sinica*, **1**, 93–125.

See Also

`amlpoisson`, `amlbinomial`, `amlexponential`, `bmi.nz`, `extlogF1`, `alaplace1`, `denorm`, `lms.bcn` and similar variants are alternative methods for quantile regression.

Examples

```
## Not run:
# Example 1
ooo <- with(bmi.nz, order(age))
bmi.nz <- bmi.nz[ooo, ]  # Sort by age
(fit <- vglm(BMI ~ sm.bs(age), amlnormal(w.aml = 0.1), bmi.nz))
fit@extra  # Gives the w value and the percentile
coef(fit, matrix = TRUE)

# Quantile plot
with(bmi.nz, plot(age, BMI, col = "blue", main =
     paste(round(fit@extra$percentile, digits = 1),
           "expectile-percentile curve")))
with(bmi.nz, lines(age, c(fitted(fit)), col = "black"))

# Example 2
# Find the w values that give the 25, 50 and 75 percentiles
find.w <- function(w, percentile = 50) {
  fit2 <- vglm(BMI ~ sm.bs(age), amlnormal(w = w), data = bmi.nz)
  fit2@extra$percentile - percentile
}
# Quantile plot
with(bmi.nz, plot(age, BMI, col = "blue", las = 1, main =
     "25, 50 and 75 expectile-percentile curves"))
for (myp in c(25, 50, 75)) {
# Note: uniroot() can only find one root at a time
  bestw <- uniroot(f = find.w, interval = c(1/10^4, 10^4),
                   percentile = myp)
  fit2 <- vglm(BMI ~ sm.bs(age), amlnormal(w = bestw$root), bmi.nz)
  with(bmi.nz, lines(age, c(fitted(fit2)), col = "orange"))
}
```

```
# Example 3; this is Example 1 but with smoothing splines and
# a vector w and a parallelism assumption.
ooo <- with(bmi.nz, order(age))
bmi.nz <- bmi.nz[ooo, ]  # Sort by age
fit3 <- vgam(BMI ~ s(age, df = 4), data = bmi.nz, trace = TRUE,
             amlnormal(w = c(0.1, 1, 10), parallel = TRUE))
fit3@extra  # The w values, percentiles and weighted deviances

# The linear components of the fit; not for human consumption:
coef(fit3, matrix = TRUE)

# Quantile plot
with(bmi.nz, plot(age, BMI, col="blue", main =
  paste(paste(round(fit3@extra$percentile, digits = 1), collapse = ", "),
        "expectile-percentile curves")))
with(bmi.nz, matlines(age, fitted(fit3), col = 1:fit3@extra$M, lwd = 2))
with(bmi.nz, lines(age, c(fitted(fit )), col = "black"))  # For comparison

## End(Not run)
```

---

amlpoisson *Poisson Regression by Asymmetric Maximum Likelihood Estimation*

Description

Poisson quantile regression estimated by maximizing an asymmetric likelihood function.

Usage

```
amlpoisson(w.aml = 1, parallel = FALSE, imethod = 1, digw = 4,
           link = "loglink")
```

Arguments

`w.aml`	Numeric, a vector of positive constants controlling the percentiles. The larger the value the larger the fitted percentile value (the proportion of points below the "w-regression plane"). The default value of unity results in the ordinary maximum likelihood (MLE) solution.
`parallel`	If `w.aml` has more than one value then this argument allows the quantile curves to differ by the same amount as a function of the covariates. Setting this to be `TRUE` should force the quantile curves to not cross (although they may not cross anyway). See `CommonVGAMffArguments` for more information.
`imethod`	Integer, either 1 or 2 or 3. Initialization method. Choose another value if convergence fails.
`digw`	Passed into `Round` as the `digits` argument for the `w.aml` values; used cosmetically for labelling.
`link`	See `poissonff`.

Details

This method was proposed by Efron (1992) and full details can be obtained there.

The model is essentially a Poisson regression model (see `poissonff`) but the usual deviance is replaced by an asymmetric squared error loss function; it is multiplied by $w.aml$ for positive residuals. The solution is the set of regression coefficients that minimize the sum of these deviance-type values over the data set, weighted by the `weights` argument (so that it can contain frequencies). Newton-Raphson estimation is used here.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Warning

If `w.aml` has more than one value then the value returned by `deviance` is the sum of all the (weighted) deviances taken over all the `w.aml` values. See Equation (1.6) of Efron (1992).

Note

On fitting, the `extra` slot has list components `"w.aml"` and `"percentile"`. The latter is the percent of observations below the “w-regression plane”, which is the fitted values. Also, the individual deviance values corresponding to each element of the argument `w.aml` is stored in the `extra` slot.

For `amlpoisson` objects, methods functions for the generic functions `qtplot` and `cdf` have not been written yet.

About the jargon, Newey and Powell (1987) used the name *expectiles* for regression surfaces obtained by asymmetric least squares. This was deliberate so as to distinguish them from the original *regression quantiles* of Koenker and Bassett (1978). Efron (1991) and Efron (1992) use the general name *regression percentile* to apply to all forms of asymmetric fitting. Although the asymmetric maximum likelihood method very nearly gives regression percentiles in the strictest sense for the normal and Poisson cases, the phrase *quantile regression* is used loosely in this **VGAM** documentation.

In this documentation the word *quantile* can often be interchangeably replaced by *expectile* (things are informal here).

Author(s)

Thomas W. Yee

References

Efron, B. (1991). Regression percentiles using asymmetric squared error loss. *Statistica Sinica*, **1**, 93–125.

Efron, B. (1992). Poisson overdispersion estimates based on the method of asymmetric maximum likelihood. *Journal of the American Statistical Association*, **87**, 98–107.

Koenker, R. and Bassett, G. (1978). Regression quantiles. *Econometrica*, **46**, 33–50.

Newey, W. K. and Powell, J. L. (1987). Asymmetric least squares estimation and testing. *Econometrica*, **55**, 819–847.

See Also

amlnormal, amlbinomial, extlogF1, alaplace1.

Examples

```
set.seed(1234)
mydat <- data.frame(x = sort(runif(nn <- 200)))
mydat <- transform(mydat, y = rpois(nn, exp(0 - sin(8*x))))
(fit <- vgam(y ~ s(x), fam = amlpoisson(w.aml = c(0.02, 0.2, 1, 5, 50)),
             mydat, trace = TRUE))
fit@extra

## Not run:
# Quantile plot
with(mydat, plot(x, jitter(y), col = "blue", las = 1, main =
     paste(paste(round(fit@extra$percentile, digits = 1), collapse = ", "),
           "percentile-expectile curves")))
with(mydat, matlines(x, fitted(fit), lwd = 2))
## End(Not run)
```

---

anova.vglm *Analysis of Deviance for Vector Generalized Linear Model Fits*

Description

Compute an analysis of deviance table for one or more vector generalized linear model fits.

Usage

```
## S3 method for class 'vglm'
anova(object, ..., type = c("II", "I", "III", 2, 1, 3),
     test = c("LRT", "none"), trydev = TRUE, silent = TRUE)
```

Arguments

object, ...	objects of class vglm, typically the result of a call to vglm, or a list of objects for the "vglmlist" method. Each model must have an intercept term. If "vglmlist" is used then type = 1 or type = "I" must be specified.
type	character or numeric; any one of the (effectively three) choices given. Note that anova.glm has 1 or "I" as its default; and that Anova.glm() in **car** (that is, the **car** package) has 2 or "II" as its default (and allows for type = "III"), so one can think of this function as a combination of anova.glm and Anova.glm() in **car**, but with the default of the latter. See Details below for more information.
test	a character string, (partially) matching one of "LRT" and "none". In the future it is hoped that "Rao" be also supported, to conduct score tests. The first value is the default.

`trydev` logical; if `TRUE` then the deviance is used if possible. Note that only a few **VGAM** family functions have a deviance that is defined and implemented. Setting it `FALSE` means the log-likelihood will be used.

`silent` logical; if `TRUE` then any warnings will be suppressed. These may arise by IRLS iterations not converging during the fitting of submodels. Setting it `FALSE` means that any warnings are given.

Details

`anova.vglm` is intended to be similar to `anova.glm` so specifying a single object and `type = 1` gives a *sequential* analysis of deviance table for that fit. By *analysis of deviance*, it is meant loosely that if the deviance of the model is not defined or implemented, then twice the difference between the log-likelihoods of two nested models remains asymptotically chi-squared distributed with degrees of freedom equal to the difference in the number of parameters of the two models. Of course, the usual regularity conditions are assumed to hold. For Type I, the analysis of deviance table has the reductions in the residual deviance as each term of the formula is added in turn are given in as the rows of a table, plus the residual deviances themselves. *Type I* or sequential tests (as in `anova.glm`). are computationally the easiest of the three methods. For this, the order of the terms is important, and the each term is added sequentially from first to last.

The `Anova()` function in **car** allows for testing *Type II* and *Type III* (SAS jargon) hypothesis tests, although the definitions used are *not* precisely that of SAS. As **car** notes, *Type I* rarely test interesting hypotheses in unbalanced designs. Type III enter each term *last*, keeping all the other terms in the model.

Type II tests, according to SAS, add the term after all other terms have been added to the model except terms that contain the effect being tested; an effect is contained in another effect if it can be derived by deleting variables from the latter effect. Type II tests are currently the default.

As in `anova.glm`, but not as `Anova.glm()` in **car**, if more than one object is specified, then the table has a row for the residual degrees of freedom and deviance for each model. For all but the first model, the change in degrees of freedom and deviance is also given. (This only makes statistical sense if the models are nested.) It is conventional to list the models from smallest to largest, but this is up to the user. It is necessary to have `type = 1` with more than one objects are specified.

See `anova.glm` for more details and warnings. The **VGAM** package now implements full likelihood models only, therefore no dispersion parameters are estimated.

Value

An object of class `"anova"` inheriting from class `"data.frame"`.

Warning

See `anova.glm`. Several **VGAM** family functions implement distributions which do not satisfying the usual regularity conditions needed for the LRT to work. No checking or warning is given for these.

As **car** says, be careful of Type III tests because they violate marginality. Type II tests (the default) do not have this problem.

Note

It is possible for this function to stop when type = 2 or 3, e.g., anova(vglm(cans ~ myfactor, poissonff, data = boxcar)) where myfactor is a factor.

The code was adapted directly from anova.glm and Anova.glm() in **car** by T. W. Yee. Hence the Type II and Type III tests do *not* correspond precisely with the SAS definition.

See Also

anova.glm, stat.anova, stats:::print.anova, Anova.glm() in **car** if **car** is installed, vglm, lrtest, add1.vglm, drop1.vglm, lrt.stat.vlm, score.stat.vlm, wald.stat.vlm, backPain2, update.

Examples

```
# Example 1: a proportional odds model fitted to pneumo.
set.seed(1)
pneumo <- transform(pneumo, let = log(exposure.time), x3 = runif(8))
fit1 <- vglm(cbind(normal, mild, severe) ~ let     , propodds, pneumo)
fit2 <- vglm(cbind(normal, mild, severe) ~ let + x3, propodds, pneumo)
fit3 <- vglm(cbind(normal, mild, severe) ~ let + x3, cumulative, pneumo)
anova(fit1, fit2, fit3, type = 1)  # Remember to specify 'type'!!
anova(fit2)
anova(fit2, type = "I")
anova(fit2, type = "III")

# Example 2: a proportional odds model fitted to backPain2.
data("backPain2", package = "VGAM")
summary(backPain2)
fitlogit <- vglm(pain ~ x2 * x3 * x4, propodds, data = backPain2)
coef(fitlogit)
anova(fitlogit)
anova(fitlogit, type = "I")
anova(fitlogit, type = "III")
```

AR1 *Autoregressive Process with Order-1 Family Function*

Description

Maximum likelihood estimation of the three-parameter AR-1 model

Usage

```
AR1(ldrift = "identitylink", lsd  = "loglink", lvar = "loglink", lrho = "rhobitlink",
    idrift  = NULL, isd  = NULL, ivar = NULL, irho = NULL, imethod = 1,
    ishrinkage = 0.95, type.likelihood = c("exact", "conditional"),
    type.EIM  = c("exact", "approximate"), var.arg = FALSE, nodrift = FALSE,
    print.EIM = FALSE, zero = c(if (var.arg) "var" else "sd", "rho"))
```

Arguments

`ldrift, lsd, lvar, lrho`
: Link functions applied to the scaled mean, standard deviation or variance, and correlation parameters. The parameter `drift` is known as the *drift*, and it is a scaled mean. See `Links` for more choices.

`idrift, isd, ivar, irho`
: Optional initial values for the parameters. If failure to converge occurs then try different values and monitor convergence by using `trace = TRUE`. For a S-column response, these arguments can be of length S, and they are recycled by the columns first. A value `NULL` means an initial value for each response is computed internally.

`ishrinkage, imethod, zero`
: See `CommonVGAMffArguments` for more information. The default for `zero` assumes there is a drift parameter to be estimated (the default for that argument), so if a drift parameter is suppressed and there are covariates, then `zero` will need to be assigned the value 1 or 2 or `NULL`.

`var.arg`
: Same meaning as `uninormal`.

`nodrift`
: Logical, for determining whether to estimate the drift parameter. The default is to estimate it. If `TRUE`, the drift parameter is set to 0 and not estimated.

`type.EIM`
: What type of expected information matrix (EIM) is used in Fisher scoring. By default, this family function calls `AR1EIM`, which recursively computes the exact EIM for the AR process with Gaussian white noise. See Porat and Friedlander (1986) for further details on the exact EIM.

  If `type.EIM = "approximate"` then approximate expression for the EIM of Autoregressive processes is used; this approach holds when the number of observations is large enough. Succinct details about the approximate EIM are delineated at Porat and Friedlander (1987).

`print.EIM`
: Logical. If `TRUE`, then the first few EIMs are printed. Here, the result shown is the sum of each EIM.

`type.likelihood`
: What type of likelihood function is maximized. The first choice (default) is the sum of the marginal likelihood and the conditional likelihood. Choosing the conditional likelihood means that the first observation is effectively ignored (this is handled internally by setting the value of the first prior weight to be some small positive number, e.g., `1.0e-6`). See the note below.

Details

The AR-1 model implemented here has

$$Y_1 \sim N(\mu, \sigma^2/(1-\rho^2)),$$

and

$$Y_i = \mu^* + \rho Y_{i-1} + e_i,$$

where the e_i are i.i.d. Normal(0, sd = σ) random variates.

Here are a few notes: (1). A test for weak stationarity might be to verify whether $1/\rho$ lies outside the unit circle. (2). The mean of all the Y_i is $\mu^*/(1-\rho)$ and these are returned as the fitted values. (3). The correlation of all the Y_i with Y_{i-1} is ρ. (4). The default link function ensures that $-1 < \rho < 1$.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Warning

Monitoring convergence is urged, i.e., set trace = TRUE.

Moreover, if the exact EIMs are used, set print.EIM = TRUE to compare the computed exact to the approximate EIM.

Under the VGLM/VGAM approach, parameters can be modelled in terms of covariates. Particularly, if the standard deviation of the white noise is modelled in this way, then type.EIM = "exact" may certainly lead to unstable results. The reason is that white noise is a stationary process, and consequently, its variance must remain as a constant. Consequently, the use of variates to model this parameter contradicts the assumption of stationary random components to compute the exact EIMs proposed by Porat and Friedlander (1987).

To prevent convergence issues in such cases, this family function internally verifies whether the variance of the white noise remains as a constant at each Fisher scoring iteration. If this assumption is violated and type.EIM = "exact" is set, then AR1 automatically shifts to type.EIM = "approximate". Also, a warning is accordingly displayed.

Note

Multiple responses are handled. The mean is returned as the fitted values.

Author(s)

Victor Miranda (exact method) and Thomas W. Yee (approximate method).

References

Porat, B. and Friedlander, B. (1987). The Exact Cramer-Rao Bond for Gaussian Autoregressive Processes. *IEEE Transactions on Aerospace and Electronic Systems*, **AES-23(4)**, 537–542.

See Also

AR1EIM, vglm.control, dAR1, arima.sim.

Examples

```
### Example 1: using  arima.sim() to generate a 0-mean stationary time series.
nn <- 500
tsdata <- data.frame(x2 =  runif(nn))
ar.coef.1 <- rhobitlink(-1.55, inverse = TRUE)  # Approx -0.65
ar.coef.2 <- rhobitlink( 1.0, inverse = TRUE)   # Approx  0.50
set.seed(1)
tsdata  <- transform(tsdata,
              index = 1:nn,
              TS1 = arima.sim(nn, model = list(ar = ar.coef.1),
                              sd = exp(1.5)),
```

```
              TS2 = arima.sim(nn, model = list(ar = ar.coef.2),
                              sd = exp(1.0 + 1.5 * x2)))

### An autoregressive intercept--only model.   ###
### Using the exact EIM, and "nodrift = TRUE"  ###
fit1a <- vglm(TS1 ~ 1, data = tsdata, trace = TRUE,
              AR1(var.arg = FALSE, nodrift = TRUE,
                  type.EIM = "exact",
                  print.EIM = FALSE),
              crit = "coefficients")
Coef(fit1a)
summary(fit1a)

## Not run:
### Two responses. Here, the white noise standard deviation of TS2   ###
### is modelled in terms of 'x2'. Also, 'type.EIM = exact'.  ###
fit1b <- vglm(cbind(TS1, TS2) ~ x2,
              AR1(zero = NULL, nodrift = TRUE,
                  var.arg = FALSE,
                  type.EIM = "exact"),
              constraints = list("(Intercept)" = diag(4),
                                 "x2" = rbind(0, 0, 1, 0)),
              data = tsdata, trace = TRUE, crit = "coefficients")
coef(fit1b, matrix = TRUE)
summary(fit1b)

### Example 2: another stationary time series
nn     <- 500
my.rho <- rhobitlink(1.0, inverse = TRUE)
my.mu  <- 1.0
my.sd  <- exp(1)
tsdata  <- data.frame(index = 1:nn, TS3 = runif(nn))

set.seed(2)
for (ii in 2:nn)
  tsdata$TS3[ii] <- my.mu/(1 - my.rho) +
                    my.rho * tsdata$TS3[ii-1] + rnorm(1, sd = my.sd)
tsdata <- tsdata[-(1:ceiling(nn/5)), ]  # Remove the burn-in data:

### Fitting an AR(1). The exact EIMs are used.
fit2a <- vglm(TS3 ~ 1, AR1(type.likelihood = "exact",  # "conditional",
                                type.EIM = "exact"),
              data = tsdata, trace = TRUE, crit = "coefficients")

Coef(fit2a)
summary(fit2a)      # SEs are useful to know

Coef(fit2a)["rho"]    # Estimate of rho, for intercept-only models
my.rho                # The 'truth' (rho)
Coef(fit2a)["drift"]  # Estimate of drift, for intercept-only models
my.mu /(1 - my.rho)   # The 'truth' (drift)

## End(Not run)
```

AR1EIM *Computation of the Exact EIM of an Order-1 Autoregressive Process*

Description

Computation of the exact Expected Information Matrix of the Autoregressive process of order-1 (AR(1)) with Gaussian white noise and stationary random components.

Usage

```
AR1EIM(x = NULL, var.arg = NULL, p.drift = NULL,
       WNsd = NULL, ARcoeff1 = NULL, eps.porat = 1e-2)
```

Arguments

`x`	A vector of quantiles. The gaussian time series for which the EIMs are computed. If multiple time series are being analyzed, then `x` must be a matrix where each column allocates a response. That is, the number of columns (denoted as NOS) must match the number of responses.
`var.arg`	Logical. Same as with `AR1`.
`p.drift`	A numeric vector with the *scaled mean(s)* (commonly referred as *drift*) of the AR process(es) in turn. Its length matches the number of responses.
`WNsd, ARcoeff1`	Matrices. The standard deviation of the white noise, and the correlation (coefficient) of the AR(1) model, for **each** observation. That is, the dimension for each matrix is $N \times NOS$, where N is the number of observations and NOS is the number of responses. Else, these arguments are recycled.
`eps.porat`	A very small positive number to test whether the standar deviation (`WNsd`) is close enough to its value estimated in this function. See below for further details.

Details

This function implements the algorithm of Porat and Friedlander (1986) to *recursively* compute the exact expected information matrix (EIM) of Gaussian time series with stationary random components.

By default, when the VGLM/VGAM family function `AR1` is used to fit an AR(1) model via `vglm`, Fisher scoring is executed using the **approximate** EIM for the AR process. However, this model can also be fitted using the **exact** EIMs computed by `AR1EIM`.

Given N consecutive data points, $y_0, y_1, \ldots, y_{N-1}$ with probability density $f(\boldsymbol{y})$, the Porat and Friedlander algorithm calculates the EIMs $[J_{n-1}(\boldsymbol{\theta})]$, for all $1 \leq n \leq N$. This is done based on the Levinson-Durbin algorithm for computing the orthogonal polynomials of a Toeplitz matrix. In particular, for the AR(1) model, the vector of parameters to be estimated under the VGAM/VGLM approach is

$$\boldsymbol{\eta} = (\mu^*, \log(\sigma^2), rhobit(\rho)),$$

where σ^2 is the variance of the white noise and mu^* is the drift parameter (See `AR1` for further details on this).

Consequently, for each observation $n = 1, \ldots, N$, the EIM, $J_n(\boldsymbol{\theta})$, has dimension 3×3, where the diagonal elements are:

$$J_{[n,1,1]} = E[-\partial^2 \log f(\boldsymbol{y})/\partial(\mu^*)^2],$$

$$J_{[n,2,2]} = E[-\partial^2 \log f(\boldsymbol{y})/\partial(\sigma^2)^2],$$

and

$$J_{[n,3,3]} = E[-\partial^2 \log f(\boldsymbol{y})/\partial(\rho)^2].$$

As for the off-diagonal elements, one has the usual entries, i.e.,

$$J_{[n,1,2]} = J_{[n,2,1]} = E[-\partial^2 \log f(\boldsymbol{y})/\partial\sigma^2\partial\rho],$$

etc.

If `var.arg = FALSE`, then σ instead of σ^2 is estimated. Therefore, $J_{[n,2,2]}$, $J_{[n,1,2]}$, etc., are correspondingly replaced.

Once these expected values are internally computed, they are returned in an array of dimension $N \times 1 \times 6$, of the form

$$J[, 1,] = [J_{[,1,1]}, J_{[,2,2]}, J_{[,3,3]}, J_{[,1,2]}, J_{[,2,3]}, J_{[,1,3]}].$$

`AR1EIM` handles multiple time series, say NOS. If this happens, then it accordingly returns an array of dimension $N \times NOS \times 6$. Here, $J[, k,]$, for $k = 1, \ldots, NOS$, is a matrix of dimension $N \times 6$, which stores the EIMs for the k^{th}th response, as above, i.e.,

$$J[, k,] = [J_{[,1,1]}, J_{[,2,2]}, J_{[,3,3]}, \ldots],$$

the *bandwith* form, as per required by `AR1`.

Value

An array of dimension $N \times NOS \times 6$, as above.

This array stores the EIMs calculated from the joint density as a function of

$$\boldsymbol{\theta} = (\mu^*, \sigma^2, \rho).$$

Nevertheless, note that, under the VGAM/VGLM approach, the EIMs must be correspondingly calculated in terms of the linear predictors, $\boldsymbol{\eta}$.

Asymptotic behaviour of the algorithm

For large enough n, the EIMs, $J_n(\boldsymbol{\theta})$, become approximately linear in n. That is, for some n_0,

$$J_n(\boldsymbol{\theta}) \equiv J_{n_0}(\boldsymbol{\theta}) + (n - n_0)\bar{J}(\boldsymbol{\theta}), \qquad (**)$$

where $\bar{J}(\boldsymbol{\theta})$ is a constant matrix.

This relationsihip is internally considered if a proper value of n_0 is determined. Different ways can be adopted to find n_0. In `AR1EIM`, this is done by checking the difference between the internally estimated variances and the entered ones at `WNsd`. If this difference is less than `eps.porat` at some iteration, say at iteration n_0, then `AR1EIM` takes $\bar{J}(\boldsymbol{\theta})$ as the last computed increment of $J_n(\boldsymbol{\theta})$, and extraplotates $J_k(\boldsymbol{\theta})$, for all $k \geq n_0$ using $(*)$. Else, the algorithm will complete the iterations for $1 \leq n \leq N$.

Finally, note that the rate of convergence reasonably decreases if the asymptotic relationship $(*)$ is used to compute $J_k(\boldsymbol{\theta})$, $k \geq n_0$. Normally, the number of operations involved on this algorithm is proportional to N^2.

See Porat and Friedlander (1986) for full details on the asymptotic behaviour of the algorithm.

Warning

Arguments `WNsd`, and `ARcoeff1` are matrices of dimension $N \times NOS$. Else, these arguments are accordingly recycled.

Note

For simplicity, one can assume that the time series analyzed has a 0-mean. Consequently, where the family function `AR1` calls `AR1EIM` to compute the EIMs, the argument `p.drift` is internally set to zero-vector, whereas `x` is *centered* by subtracting its mean value.

Author(s)

V. Miranda and T. W. Yee.

References

Porat, B. and Friedlander, B. (1986). Computation of the Exact Information Matrix of Gaussian Time Series with Stationary Random Components. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, **54(1)**, 118–130.

See Also

`AR1`.

Examples

```
set.seed(1)
nn <- 500
ARcoeff1 <- c(0.3, 0.25)        # Will be recycled.
WNsd     <- c(exp(1), exp(1.5)) # Will be recycled.
p.drift  <- c(0, 0)             # Zero-mean gaussian time series.
```

```
### Generate two (zero-mean) AR(1) processes ###
ts1 <- p.drift[1]/(1 - ARcoeff1[1]) +
                 arima.sim(model = list(ar = ARcoeff1[1]), n = nn,
                 sd = WNsd[1])
ts2 <- p.drift[2]/(1 - ARcoeff1[2]) +
                 arima.sim(model = list(ar = ARcoeff1[2]), n = nn,
                 sd = WNsd[2])

ARdata <- matrix(cbind(ts1, ts2), ncol = 2)


### Compute the exact EIMs: TWO responses. ###
ExactEIM <- AR1EIM(x = ARdata, var.arg = FALSE, p.drift = p.drift,
                         WNsd = WNsd, ARcoeff1 = ARcoeff1)

### For response 1:
head(ExactEIM[, 1 ,])      # NOTICE THAT THIS IS A (nn x 6) MATRIX!

### For response 2:
head(ExactEIM[, 2 ,])      # NOTICE THAT THIS IS A (nn x 6) MATRIX!
```

---

asinlink *Arcsine Link Function*

Description

Computes the arcsine link, including its inverse and the first few derivatives.

Usage

```
asinlink(theta, bvalue = NULL, inverse = FALSE,
   deriv = 0, short = TRUE, tag = FALSE, c10 = c(4, -pi))
```

Arguments

`theta`	Numeric or character. See below for further details.
`bvalue`	See `Links`.
`inverse, deriv, short, tag`	Details at `Links`.
`c10`	Similar to `sqrtlink`. The default is intended to match `lcalogitlink` for `binomialff` at binomial probabilities (`theta`) equal to 0.5.

Details

Function `alogitlink` gives some motivation for this link. However, the problem with this link is that it is bounded by default between `(-pi, pi)` so that it can be unsuitable for regression. This link is a scaled and centred CDF of the arcsine distribution. The centring is chosen so that `asinlink(0.5)` is 0, and the scaling is chosen so that `asinlink(0.5, deriv = 1)` and `logitlink(0.5,` deriv = 1) are equal (the value 4 actually), hence this link will operate similar to the logitlink when close to 0.5.

Value

Similar to logitlink but using different formulas.

Warning

It is possible that the scaling might change in the future.

Author(s)

Thomas W. Yee

See Also

logitlink, alogitlink, Links, probitlink, clogloglink, cauchitlink, binomialff, sloglink, hdeff.

Examples

```
p <- seq(0.01, 0.99, length= 10)
asinlink(p)
max(abs(asinlink(asinlink(p), inv = TRUE) - p))  # 0?

## Not run:
par(mfrow = c(2, 2), lwd = (mylwd <- 2))
y <- seq(-4, 4, length = 100)
p <- seq(0.01, 0.99, by = 0.01)

for (d in 0:1) {
  matplot(p, cbind(logitlink(p, deriv = d), probitlink(p, deriv = d)),
          type = "n", col = "blue", ylab = "transformation",
          log = ifelse(d == 1, "y", ""),
          las = 1, main = if (d == 0) "Some probability link functions"
          else "First derivative")
  lines(p,   logitlink(p, deriv = d), col = "green")
  lines(p,  probitlink(p, deriv = d), col = "blue")
  lines(p, clogloglink(p, deriv = d), col = "tan")
  lines(p,    asinlink(p, deriv = d), col = "red3")
  if (d == 0) {
    abline(v = 0.5, h = 0, lty = "dashed")
    legend(0, 4.5, c("logitlink", "probitlink", "clogloglink",
           "asinlink"), lwd = mylwd,
           col = c("green", "blue", "tan", "red3"))
  } else
    abline(v = 0.5, lwd = 0.5, col = "gray")
}

for (d in 0) {
  matplot(y, cbind( logitlink(y, deriv = d, inverse = TRUE),
```

```
                 probitlink(y, deriv = d, inverse = TRUE)),
        type  = "n", col = "blue", xlab = "transformation", ylab = "p",
        main = if (d == 0) "Some inverse probability link functions"
        else "First derivative", las=1)
  lines(y,   logitlink(y, deriv = d, inverse = TRUE), col = "green")
  lines(y,  probitlink(y, deriv = d, inverse = TRUE), col = "blue")
  lines(y, clogloglink(y, deriv = d, inverse = TRUE), col = "tan")
  lines(y,    asinlink(y, deriv = d, inverse = TRUE), col = "red3")
  if (d == 0) {
      abline(h = 0.5, v = 0, lwd = 0.5, col = "gray")
      legend(-4, 1, c("logitlink", "probitlink", "clogloglink",
             "asinlink"), lwd = mylwd,
             col = c("green", "blue", "tan", "red3"))
  }
}
par(lwd = 1)

## End(Not run)
```

---

auuc — *Auckland University Undergraduate Counts Data*

Description

Undergraduate student enrolments at the University of Auckland in 1990.

Usage

```
data(auuc)
```

Format

A data frame with 4 observations on the following 5 variables.

Commerce a numeric vector of counts.

Arts a numeric vector of counts.

SciEng a numeric vector of counts.

Law a numeric vector of counts.

Medicine a numeric vector of counts.

Details

Each student is cross-classified by their colleges (Science and Engineering have been combined) and the socio-economic status (SES) of their fathers (1 = highest, down to 4 = lowest).

Source

Dr Tony Morrison.

References

Wild, C. J. and Seber, G. A. F. (2000). *Chance Encounters: A First Course in Data Analysis and Inference*, New York: Wiley.

Examples

```
auuc
## Not run:
round(fitted(grc(auuc)))
round(fitted(grc(auuc, Rank = 2)))

## End(Not run)
```

---

aux.posbernoulli.t — *Auxiliary Function for the Positive Bernoulli Family Function with Time Effects*

Description

Returns behavioural effects indicator variables from a capture history matrix.

Usage

```
aux.posbernoulli.t(y, check.y = FALSE, rename = TRUE, name = "bei")
```

Arguments

`y`	Capture history matrix. Rows are animals, columns are sampling occasions, and values should be 0s and 1s only.
`check.y`	Logical, if `TRUE` then some basic checking is performed.
`rename, name`	If `rename = TRUE` then the behavioural effects indicator are named using the value of `name` as the prefix. If `FALSE` then use the same column names as `y`.

Details

This function can help fit certain capture–recapture models (commonly known as M_{tb} or M_{tbh} (no prefix h means it is an intercept-only model) in the literature). See `posbernoulli.t` for details.

Value

A list with the following components.

cap.hist1 A matrix the same dimension as `y`. In any particular row there are 0s up to the first capture. Then there are 1s thereafter.

cap1 A vector specifying which time occasion the animal was first captured.

y0i Number of noncaptures before the first capture.

yr0i Number of noncaptures after the first capture.

yr1i Number of recaptures after the first capture.

See Also

posbernoulli.t, deermice.

Examples

```
# Fit a M_tbh model to the deermice data:
(pdata <- aux.posbernoulli.t(with(deermice,
                                  cbind(y1, y2, y3, y4, y5, y6))))

deermice <- data.frame(deermice,
                   bei = 0,  # Add this
                   pdata$cap.hist1)  # Incorporate these
head(deermice)  # Augmented with behavioural effect indicator variables
tail(deermice)
```

---

backPain *Data on Back Pain Prognosis, from Anderson (1984)*

Description

Data from a study of patients suffering from back pain. Prognostic variables were recorded at presentation and progress was categorised three weeks after treatment.

Usage

```
data(backPain)
```

Format

A data frame with 101 observations on the following 4 variables.

x2 length of previous attack.

x3 pain change.

x4 lordosis.

pain an ordered factor describing the progress of each patient with levels `worse` < `same` < `slight.improvement` < `moderate.improvement` < `marked.improvement` < `complete.relief`.

Source

http://ideas.repec.org/c/boc/bocode/s419001.html

The data set and this help file was copied from **gnm** so that a vignette in **VGAM** could be run; the analysis is described in Yee (2010).

The data frame `backPain2` is a modification of `backPain` where the variables have been renamed (`x1` becomes `x2`, `x2` becomes `x3`, `x3` becomes `x4`) and converted into factors.

References

Anderson, J. A. (1984). Regression and Ordered Categorical Variables. *J. R. Statist. Soc. B*, **46(1)**, 1-30.

Yee, T. W. (2010). The **VGAM** package for categorical data analysis. *Journal of Statistical Software*, **32**, 1–34. doi:10.18637/jss.v032.i10.

Examples

```
summary(backPain)
summary(backPain2)
```

`beggs` *Bacon and Eggs Data*

Description

Purchasing of bacon and eggs.

Usage

```
data(beggs)
```

Format

Data frame of a two way table.

b0, b1, b2, b3, b4 The `b` refers to bacon. The number of times bacon was purchased was 0, 1, 2, 3, or 4.

e0, e1, e2, e3, e4 The `e` refers to eggs. The number of times eggs was purchased was 0, 1, 2, 3, or 4.

Details

The data is from Information Resources, Inc., a consumer panel based in a large US city [see Bell and Lattin (1998) for further details]. Starting in June 1991, the purchases in the bacon and fresh eggs product categories for a sample of 548 households over four consecutive store trips was tracked. Only those grocery shopping trips with a total basket value of at least five dollars was considered. For each household, the total number of bacon purchases in their four eligible shopping trips and the total number of egg purchases (usually a package of eggs) for the same trips, were counted.

Source

Bell, D. R. and Lattin, J. M. (1998) Shopping Behavior and Consumer Preference for Store Price Format: Why 'Large Basket' Shoppers Prefer EDLP. *Marketing Science*, **17**, 66–88.

References

Danaher, P. J. and Hardie, B. G. S. (2005). Bacon with Your Eggs? Applications of a New Bivariate Beta-Binomial Distribution. *American Statistician*, **59**(4), 282–286.

See Also

`rrvglm`, `rcim`, `grc`.

Examples

```
beggs
colSums(beggs)
rowSums(beggs)
```

`bell` *The Bell Series of Integers*

Description

Returns the values of the Bell series.

Usage

```
bell(n)
```

Arguments

`n`	Vector of non-negative integers. Values greater than 218 return an `Inf`. Non-integers or negative values return a `NaN`.

Details

The Bell numbers emerge from a series expansion of $\exp(e^x - 1)$ for real x. The first few values are $B_0 = 1$, $B_1 = 1$, $B_2 = 2$, $B_3 = 5$, $B_4 = 15$. The series increases quickly so that overflow occurs when its argument is more than 218.

Value

This function returns B_n.

Author(s)

T. W. Yee

References

Bell, E. T. (1934). Exponential polynomials. *Ann. Math.*, **35**, 258–277.

Bell, E. T. (1934). Exponential numbers. *Amer. Math. Monthly*, **41**, 411–419.

See Also

bellff, rbell.

Examples

```
 ## Not run:
plot(0:10, bell(0:10), log = "y", type = "h", col = "blue")

## End(Not run)
```

Benford *Benford's Distribution*

Description

Density, distribution function, quantile function, and random generation for Benford's distribution.

Usage

```
dbenf(x, ndigits = 1, log = FALSE)
pbenf(q, ndigits = 1, lower.tail = TRUE, log.p = FALSE)
qbenf(p, ndigits = 1, lower.tail = TRUE, log.p = FALSE)
rbenf(n, ndigits = 1)
```

Arguments

`x, q`	Vector of quantiles. See `ndigits`.
`p`	vector of probabilities.
`n`	number of observations. A single positive integer. Else if `length(n) > 1` then the length is taken to be the number required.
`ndigits`	Number of leading digits, either 1 or 2. If 1 then the support of the distribution is {1,...,9}, else {10,...,99}.
`log, log.p`	Logical. If `log.p = TRUE` then all probabilities `p` are given as `log(p)`.
`lower.tail`	Same meaning as in `pnorm` or `qnorm`.

Details

Benford's Law (aka *the significant-digit law*) is the empirical observation that in many naturally occuring tables of numerical data, the leading significant (nonzero) digit is not uniformly distributed in $\{1, 2, \ldots, 9\}$. Instead, the leading significant digit ($= D$, say) obeys the law

$$P(D = d) = \log_{10}\left(1 + \frac{1}{d}\right)$$

for $d = 1, \ldots, 9$. This means the probability the first significant digit is 1 is approximately 0.301, etc.

Benford's Law was apparently first discovered in 1881 by astronomer/mathematician S. Newcombe. It started by the observation that the pages of a book of logarithms were dirtiest at the beginning and progressively cleaner throughout. In 1938, a General Electric physicist called F. Benford rediscovered the law on this same observation. Over several years he collected data from different sources as different as atomic weights, baseball statistics, numerical data from *Reader's Digest*, and drainage areas of rivers.

Applications of Benford's Law has been as diverse as to the area of fraud detection in accounting and the design computers.

Benford's distribution has been called "a" logarithmic distribution; see `logff`.

Value

`dbenf` gives the density, `pbenf` gives the distribution function, and `qbenf` gives the quantile function, and `rbenf` generates random deviates.

Author(s)

T. W. Yee and Kai Huang

References

Benford, F. (1938). The Law of Anomalous Numbers. *Proceedings of the American Philosophical Society*, **78**, 551–572.

Newcomb, S. (1881). Note on the Frequency of Use of the Different Digits in Natural Numbers. *American Journal of Mathematics*, **4**, 39–40.

Examples

```
dbenf(x <- c(0:10, NA, NaN, -Inf, Inf))
pbenf(x)

## Not run:
xx <- 1:9
barplot(dbenf(xx), col = "lightblue", xlab = "Leading digit",
        ylab = "Probability", names.arg = as.character(xx),
        main = "Benford's distribution", las = 1)

hist(rbenf(1000), border = "blue", prob = TRUE,
     main = "1000 random variates from Benford's distribution",
     xlab = "Leading digit", sub="Red is the true probability",
     breaks = 0:9 + 0.5, ylim = c(0, 0.35), xlim = c(0, 10.0))
lines(xx, dbenf(xx), col = "red", type = "h")
points(xx, dbenf(xx), col = "red")

## End(Not run)
```

Benini *The Benini Distribution*

Description

Density, distribution function, quantile function and random generation for the Benini distribution with parameter shape.

Usage

```
dbenini(x, y0, shape, log = FALSE)
pbenini(q, y0, shape, lower.tail = TRUE, log.p = FALSE)
qbenini(p, y0, shape, lower.tail = TRUE, log.p = FALSE)
rbenini(n, y0, shape)
```

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
n	number of observations. Same as runif.
y0	the scale parameter y_0.
shape	the positive shape parameter b.
log	Logical. If log = TRUE then the logarithm of the density is returned.
lower.tail, log.p	Same meaning as in pnorm or qnorm.

Details

See benini1, the **VGAM** family function for estimating the parameter s by maximum likelihood estimation, for the formula of the probability density function and other details.

Value

dbenini gives the density, pbenini gives the distribution function, qbenini gives the quantile function, and rbenini generates random deviates.

Author(s)

T. W. Yee and Kai Huang

References

Kleiber, C. and Kotz, S. (2003). *Statistical Size Distributions in Economics and Actuarial Sciences*, Hoboken, NJ, USA: Wiley-Interscience.

See Also

benini1.

Examples

```
## Not run:
y0 <- 1; shape <- exp(1)
xx <- seq(0.0, 4, len = 101)
plot(xx, dbenini(xx, y0 = y0, shape = shape), col = "blue",
     main = "Blue is density, orange is the CDF", type = "l",
     sub = "Purple lines are the 10,20,...,90 percentiles",
     ylim = 0:1, las = 1, ylab = "", xlab = "x")
abline(h = 0, col = "blue", lty = 2)
lines(xx, pbenini(xx, y0 = y0, shape = shape), col = "orange")
probs <- seq(0.1, 0.9, by = 0.1)
Q <- qbenini(probs, y0 = y0, shape = shape)
lines(Q, dbenini(Q, y0 = y0, shape = shape),
      col = "purple", lty = 3, type = "h")
pbenini(Q, y0 = y0, shape = shape) - probs  # Should be all zero

## End(Not run)
```

---

benini1 *Benini Distribution Family Function*

Description

Estimating the 1-parameter Benini distribution by maximum likelihood estimation.

Usage

```
benini1(y0 = stop("argument 'y0' must be specified"),
        lshape = "loglink", ishape = NULL, imethod = 1,
        zero = NULL, parallel = FALSE,
        type.fitted = c("percentiles", "Qlink"),
        percentiles = 50)
```

Arguments

`y0` Positive scale parameter.

`lshape` Parameter link function and extra argument of the parameter b, which is the shape parameter. See Links for more choices. A log link is the default because b is positive.

`ishape` Optional initial value for the shape parameter. The default is to compute the value internally.

`imethod, zero, parallel`
Details at CommonVGAMffArguments.

`type.fitted, percentiles`
: See `CommonVGAMffArguments` for information. Using `"Qlink"` is for quantile-links in **VGAMextra**.

Details

The Benini distribution has a probability density function that can be written

$$f(y) = 2s \exp(-s[(\log(y/y_0))^2]) \log(y/y_0)/y$$

for $0 < y_0 < y$, and shape $s > 0$. The cumulative distribution function for Y is

$$F(y) = 1 - \exp(-s[(\log(y/y_0))^2]).$$

Here, Newton-Raphson and Fisher scoring coincide. The median of Y is now returned as the fitted values, by default. This **VGAM** family function can handle a multiple responses, which is inputted as a matrix.

On fitting, the `extra` slot has a component called `y0` which contains the value of the `y0` argument.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Note

Yet to do: the 2-parameter Benini distribution estimates another shape parameter a too. Hence, the code may change in the future.

Author(s)

T. W. Yee

References

Kleiber, C. and Kotz, S. (2003). *Statistical Size Distributions in Economics and Actuarial Sciences*, Hoboken, NJ, USA: Wiley-Interscience.

See Also

`Benini`.

Examples

```
y0 <- 1; nn <- 3000
bdata <- data.frame(y  = rbenini(nn, y0 = y0, shape = exp(2)))
fit <- vglm(y ~ 1, benini1(y0 = y0), data = bdata, trace = TRUE)
coef(fit, matrix = TRUE)
Coef(fit)
fit@extra$y0
c(head(fitted(fit), 1), with(bdata, median(y)))  # Should be equal
```

Betabinom *The Beta-Binomial Distribution*

Description

Density, distribution function, and random generation for the beta-binomial distribution and the inflated beta-binomial distribution.

Usage

```
dbetabinom(x, size, prob, rho = 0, log = FALSE)
pbetabinom(q, size, prob, rho = 0, log.p = FALSE)
rbetabinom(n, size, prob, rho = 0)
dbetabinom.ab(x, size, shape1, shape2, log = FALSE,
              Inf.shape = exp(20), limit.prob = 0.5)
pbetabinom.ab(q, size, shape1, shape2, limit.prob = 0.5,
              log.p = FALSE)
rbetabinom.ab(n, size, shape1, shape2, limit.prob = 0.5,
              .dontuse.prob = NULL)
dzoibetabinom(x, size, prob, rho = 0, pstr0 = 0, pstrsize = 0,
              log = FALSE)
pzoibetabinom(q, size, prob, rho, pstr0 = 0, pstrsize = 0,
              lower.tail = TRUE, log.p = FALSE)
rzoibetabinom(n, size, prob, rho = 0, pstr0 = 0, pstrsize = 0)
dzoibetabinom.ab(x, size, shape1, shape2, pstr0 = 0, pstrsize = 0,
                 log = FALSE)
pzoibetabinom.ab(q, size, shape1, shape2, pstr0 = 0, pstrsize = 0,
              lower.tail = TRUE, log.p = FALSE)
rzoibetabinom.ab(n, size, shape1, shape2, pstr0 = 0, pstrsize = 0)
```

Arguments

`x, q`	vector of quantiles.
`size`	number of trials.
`n`	number of observations. Same as `runif`.
`prob`	the probability of success μ. Must be in the unit closed interval $[0, 1]$.
`rho`	the correlation parameter ρ, which should be in the interval $[0, 1)$. The default value of 0 corresponds to the usual binomial distribution with probability `prob`. Setting `rho = 1` would set both shape parameters equal to 0, and the ratio `0/0`, which is actually `NaN`, is interpreted by `Beta` as 0.5. See the warning below.
`shape1, shape2`	the two (positive) shape parameters of the standard beta distribution. They are called `a` and `b` in `beta` respectively. Note that `shape1 = prob*(1-rho)/rho` and `shape2 = (1-prob)*(1-rho)/rho` is an important relationship between the parameters, so that the shape parameters are infinite by default because `rho = 0`; hence `limit.prob = prob` is used to obtain the behaviour of the usual binomial distribution.

log, log.p, lower.tail
: Same meaning as runif.

Inf.shape
: Numeric. A large value such that, if shape1 or shape2 exceeds this, then special measures are taken, e.g., calling dbinom. Also, if shape1 or shape2 is less than its reciprocal, then special measures are also taken. This feature/approximation is needed to avoid numerical problem with catastrophic cancellation of multiple lbeta calls.

limit.prob
: Numerical vector; recycled if necessary. If either shape parameters are Inf then the binomial limit is taken, with shape1 / (shape1 + shape2) as the probability of success. In the case where both are Inf this probability will be a NaN = Inf/Inf, however, the value limit.prob is used instead. Hence the default for dbetabinom.ab() is to assume that both shape parameters are equal as the limit is taken (indeed, Beta uses 0.5). Note that for [dpr]betabinom(), because rho = 0 by default, then limit.prob = prob so that the beta-binomial distribution behaves like the ordinary binomial distribution with respect to arguments size and prob.

.dontuse.prob
: An argument that should be ignored and *not* used.

pstr0
: Probability of a structual zero (i.e., ignoring the beta-binomial distribution). The default value of pstr0 corresponds to the response having a beta-binomial distribuion inflated only at size.

pstrsize
: Probability of a structual maximum value size. The default value of pstrsize corresponds to the response having a beta-binomial distribution inflated only at 0.

Details

The beta-binomial distribution is a binomial distribution whose probability of success is not a constant but it is generated from a beta distribution with parameters shape1 and shape2. Note that the mean of this beta distribution is mu = shape1/(shape1+shape2), which therefore is the mean or the probability of success.

See betabinomial and betabinomialff, the **VGAM** family functions for estimating the parameters, for the formula of the probability density function and other details.

For the inflated beta-binomial distribution, the probability mass function is

$$P(Y = y) = (1 - pstr0 - pstrsize) \times BB(y) + pstr0 \times I[y = 0] + pstrsize \times I[y = size]$$

where $BB(y)$ is the probability mass function of the beta-binomial distribution with the same shape parameters (pbetabinom.ab), pstr0 is the inflated probability at 0 and pstrsize is the inflated probability at 1. The default values of pstr0 and pstrsize mean that these functions behave like the ordinary Betabinom when only the essential arguments are inputted.

Value

dbetabinom and dbetabinom.ab give the density, pbetabinom and pbetabinom.ab give the distribution function, and rbetabinom and rbetabinom.ab generate random deviates.

dzoibetabinom and dzoibetabinom.ab give the inflated density, pzoibetabinom and pzoibetabinom.ab give the inflated distribution function, and rzoibetabinom and rzoibetabinom.ab generate random inflated deviates.

Warning

Setting `rho = 1` is not recommended, however the code may be modified in the future to handle this special case.

Note

`pzoibetabinom`, `pzoibetabinom.ab`, `pbetabinom` and `pbetabinom.ab` can be particularly slow. The functions here ending in `.ab` are called from those functions which don't. The simple transformations $\mu = \alpha/(\alpha + \beta)$ and $\rho = 1/(1 + \alpha + \beta)$ are used, where α and β are the two shape parameters.

Author(s)

T. W. Yee and Xiangjie Xue

See Also

`Extbetabinom`, `betabinomial`, `betabinomialff`, `Zoabeta`, `Beta`.

Examples

```
set.seed(1); rbetabinom(10, 100, prob = 0.5)
set.seed(1);     rbinom(10, 100, prob = 0.5)  # The same as rho = 0

## Not run:  N <- 9; xx <- 0:N; s1 <- 2; s2 <- 3
dy <- dbetabinom.ab(xx, size = N, shape1 = s1, shape2 = s2)
barplot(rbind(dy, dbinom(xx, size = N, prob = s1 / (s1+s2))),
        beside = TRUE, col = c("blue","green"), las = 1,
        main = paste("Beta-binomial (size=",N,", shape1=", s1,
                   ", shape2=", s2, ") (blue) vs\n",
        " Binomial(size=", N, ", prob=", s1/(s1+s2), ") (green)",
                     sep = ""),
        names.arg = as.character(xx), cex.main = 0.8)
sum(dy * xx)  # Check expected values are equal
sum(dbinom(xx, size = N, prob = s1 / (s1+s2)) * xx)
# Should be all 0:
cumsum(dy) - pbetabinom.ab(xx, N, shape1 = s1, shape2 = s2)

y <- rbetabinom.ab(n = 1e4, size = N, shape1 = s1, shape2 = s2)
ty <- table(y)
barplot(rbind(dy, ty / sum(ty)),
        beside = TRUE, col = c("blue", "orange"), las = 1,
        main = paste("Beta-binomial (size=", N, ", shape1=", s1,
                     ", shape2=", s2, ") (blue) vs\n",
        " Random generated beta-binomial(size=", N, ", prob=",
        s1/(s1+s2), ") (orange)", sep = ""), cex.main = 0.8,
        names.arg = as.character(xx))

N <- 1e5; size <- 20; pstr0 <- 0.2; pstrsize <- 0.2
kk <- rzoibetabinom.ab(N, size, s1, s2, pstr0, pstrsize)
hist(kk, probability = TRUE, border = "blue", ylim = c(0, 0.25),
```

```
     main = "Blue/green = inflated; orange = ordinary beta-binomial",
     breaks = -0.5 : (size + 0.5))
sum(kk == 0) / N  # Proportion of 0
sum(kk == size) / N  # Proportion of size
lines(0 : size,
      dbetabinom.ab(0 : size, size, s1, s2), col = "orange")
lines(0 : size, col = "green", type = "b",
      dzoibetabinom.ab(0 : size, size, s1, s2, pstr0, pstrsize))

## End(Not run)
```

`betabinomial` *Beta-binomial Distribution Family Function*

Description

Fits a beta-binomial distribution by maximum likelihood estimation. The two parameters here are the mean and correlation coefficient.

Usage

```
betabinomial(lmu = "logitlink", lrho = "logitlink",
   irho = NULL, imethod = 1,
   ishrinkage = 0.95, nsimEIM = NULL, zero = "rho")
```

Arguments

`lmu, lrho` Link functions applied to the two parameters. See `Links` for more choices. The defaults ensure the parameters remain in $(0, 1)$, however, see the warning below. For `lrho`, `log1plink` (with an offset `log(size - 1)` for η_2) and `cloglink` may be very good choices.

`irho` Optional initial value for the correlation parameter. If given, it must be in $(0, 1)$, and is recyled to the necessary length. Assign this argument a value if a convergence failure occurs. Having `irho = NULL` means an initial value is obtained internally, though this can give unsatisfactory results.

`imethod` An integer with value `1` or `2` or ..., which specifies the initialization method for μ. If failure to converge occurs try the another value and/or else specify a value for `irho`.

`zero` Specifyies which linear/additive predictor is to be modelled as an intercept only. If assigned, the single value can be either `1` or `2`. The default is to have a single correlation parameter. To model both parameters as functions of the covariates assign `zero = NULL`. See `CommonVGAMffArguments` for more information.

`ishrinkage, nsimEIM`
See `CommonVGAMffArguments` for more information. The argument `ishrinkage` is used only if `imethod = 2`. Using the argument `nsimEIM` may offer large advantages for large values of N and/or large data sets.

Details

There are several parameterizations of the beta-binomial distribution. This family function directly models the mean and correlation parameter, i.e., the probability of success. The model can be written $T|P = p \sim Binomial(N, p)$ where P has a beta distribution with shape parameters α and β. Here, N is the number of trials (e.g., litter size), $T = NY$ is the number of successes, and p is the probability of a success (e.g., a malformation). That is, Y is the *proportion* of successes. Like `binomialff`, the fitted values are the estimated probability of success (i.e., $E[Y]$ and not $E[T]$) and the prior weights N are attached separately on the object in a slot.

The probability function is

$$P(T = t) = \binom{N}{t} \frac{Be(\alpha + t, \beta + N - t)}{Be(\alpha, \beta)}$$

where $t = 0, 1, \ldots, N$, and Be is the `beta` function with shape parameters α and β. Recall $Y = T/N$ is the real response being modelled.

The default model is $\eta_1 = logit(\mu)$ and $\eta_2 = logit(\rho)$ because both parameters lie between 0 and 1. The mean (of Y) is $p = \mu = \alpha/(\alpha+\beta)$ and the variance (of Y) is $\mu(1-\mu)(1+(N-1)\rho)/N$. Here, the correlation ρ is given by $1/(1+\alpha+\beta)$ and is the correlation between the N individuals within a litter. A *litter effect* is typically reflected by a positive value of ρ. It is known as the *over-dispersion parameter*.

This family function uses Fisher scoring. Elements of the second-order expected derivatives with respect to α and β are computed numerically, which may fail for large α, β, N or else take a long time.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`.

Suppose `fit` is a fitted beta-binomial model. Then `depvar(fit)` are the sample proportions y, `fitted(fit)` returns estimates of $E(Y)$, and `weights(fit, type = "prior")` returns the number of trials N.

Warning

If the estimated rho parameter is close to 0 then a good solution is to use `extbetabinomial`. Or you could try `lrho = "rhobitlink"`.

This family function is prone to numerical difficulties due to the expected information matrices not being positive-definite or ill-conditioned over some regions of the parameter space. If problems occur try setting `irho` to some numerical value, `nsimEIM = 100`, say, or else use `etastart` argument of `vglm`, etc.

Note

This function processes the input in the same way as `binomialff`. But it does not handle the case $N = 1$ very well because there are two parameters to estimate, not one, for each row of the input. Cases where $N = 1$ can be omitted via the `subset` argument of `vglm`.

The *extended* beta-binomial distribution of Prentice (1986) implemented by `extbetabinomial` is the preferred **VGAM** family function for BBD regression.

Author(s)

T. W. Yee

References

Moore, D. F. and Tsiatis, A. (1991). Robust estimation of the variance in moment methods for extra-binomial and extra-Poisson variation. *Biometrics*, **47**, 383–401.

See Also

extbetabinomial, betabinomialff, Betabinom, binomialff, betaff, dirmultinomial, log1plink, cloglink, lirat, simulate.vlm.

Examples

```
# Example 1
bdata <- data.frame(N = 10, mu = 0.5, rho = 0.8)
bdata <- transform(bdata,
            y = rbetabinom(100, size = N, prob = mu, rho = rho))
fit <- vglm(cbind(y, N-y) ~ 1, betabinomial, bdata, trace = TRUE)
coef(fit, matrix = TRUE)
Coef(fit)
head(cbind(depvar(fit), weights(fit, type = "prior")))


# Example 2
fit <- vglm(cbind(R, N-R) ~ 1, betabinomial, lirat,
            trace = TRUE, subset = N > 1)
coef(fit, matrix = TRUE)
Coef(fit)
t(fitted(fit))
t(depvar(fit))
t(weights(fit, type = "prior"))


# Example 3, which is more complicated
lirat <- transform(lirat, fgrp = factor(grp))
summary(lirat)  # Only 5 litters in group 3
fit2 <- vglm(cbind(R, N-R) ~ fgrp + hb, betabinomial(zero = 2),
             data = lirat, trace = TRUE, subset = N > 1)
coef(fit2, matrix = TRUE)
## Not run:  with(lirat, plot(hb[N > 1], fit2@misc$rho,
         xlab = "Hemoglobin", ylab = "Estimated rho",
         pch = as.character(grp[N > 1]), col = grp[N > 1]))
## End(Not run)
## Not run:   # cf. Figure 3 of Moore and Tsiatis (1991)
with(lirat, plot(hb, R / N, pch = as.character(grp), col = grp,
         xlab = "Hemoglobin level", ylab = "Proportion Dead",
         main = "Fitted values (lines)", las = 1))
smalldf <- with(lirat, lirat[N > 1, ])
for (gp in 1:4) {
  xx <- with(smalldf, hb[grp == gp])
```

```
  yy <- with(smalldf, fitted(fit2)[grp == gp])
  ooo <- order(xx)
  lines(xx[ooo], yy[ooo], col = gp, lwd = 2)
}
## End(Not run)
```

---

betabinomialff *Beta-binomial Distribution Family Function*

Description

Fits a beta-binomial distribution by maximum likelihood estimation. The two parameters here are the shape parameters of the underlying beta distribution.

Usage

```
betabinomialff(lshape1 = "loglink", lshape2 = "loglink",
   ishape1 = 1, ishape2 = NULL, imethod = 1, ishrinkage = 0.95,
   nsimEIM = NULL, zero = NULL)
```

Arguments

`lshape1, lshape2`
: Link functions for the two (positive) shape parameters of the beta distribution. See `Links` for more choices.

`ishape1, ishape2`
: Initial value for the shape parameters. The first must be positive, and is recyled to the necessary length. The second is optional. If a failure to converge occurs, try assigning a different value to `ishape1` and/or using `ishape2`.

`zero`
: Can be an integer specifying which linear/additive predictor is to be modelled as an intercept only. If assigned, the single value should be either `1` or `2`. The default is to model both shape parameters as functions of the covariates. If a failure to converge occurs, try `zero = 2`. See `CommonVGAMffArguments` for more information.

`ishrinkage, nsimEIM, imethod`
: See `CommonVGAMffArguments` for more information. The argument `ishrinkage` is used only if `imethod = 2`. Using the argument `nsimEIM` may offer large advantages for large values of N and/or large data sets.

Details

There are several parameterizations of the beta-binomial distribution. This family function directly models the two shape parameters of the associated beta distribution rather than the probability of success (however, see **Note** below). The model can be written $T|P = p \sim Binomial(N, p)$ where P has a beta distribution with shape parameters α and β. Here, N is the number of trials (e.g., litter size), $T = NY$ is the number of successes, and p is the probability of a success (e.g., a malformation). That is, Y is the *proportion* of successes. Like `binomialff`, the fitted values are the estimated probability of success (i.e., $E[Y]$ and not $E[T]$) and the prior weights N are attached separately on the object in a slot.

The probability function is

$$P(T = t) = \binom{N}{t} \frac{B(\alpha + t, \beta + N - t)}{B(\alpha, \beta)}$$

where $t = 0, 1, \ldots, N$, and B is the beta function with shape parameters α and β. Recall $Y = T/N$ is the real response being modelled.

The default model is $\eta_1 = \log(\alpha)$ and $\eta_2 = \log(\beta)$ because both parameters are positive. The mean (of Y) is $p = \mu = \alpha/(\alpha + \beta)$ and the variance (of Y) is $\mu(1 - \mu)(1 + (N - 1)\rho)/N$. Here, the correlation ρ is given by $1/(1 + \alpha + \beta)$ and is the correlation between the N individuals within a litter. A *litter effect* is typically reflected by a positive value of ρ. It is known as the *over-dispersion parameter*.

This family function uses Fisher scoring. The two diagonal elements of the second-order expected derivatives with respect to α and β are computed numerically, which may fail for large α, β, N or else take a long time.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`.

Suppose `fit` is a fitted beta-binomial model. Then `fit@y` (better: `depvar(fit)`) contains the sample proportions y, `fitted(fit)` returns estimates of $E(Y)$, and `weights(fit, type = "prior")` returns the number of trials N.

Warning

This family function is prone to numerical difficulties due to the expected information matrices not being positive-definite or ill-conditioned over some regions of the parameter space. If problems occur try setting `ishape1` to be some other positive value, using `ishape2` and/or setting `zero = 2`.

This family function may be renamed in the future. See the warnings in `betabinomial`.

Note

This function processes the input in the same way as `binomialff`. But it does not handle the case $N = 1$ very well because there are two parameters to estimate, not one, for each row of the input. Cases where $N = 1$ can be omitted via the `subset` argument of `vglm`.

Although the two linear/additive predictors given above are in terms of α and β, basic algebra shows that the default amounts to fitting a logit link to the probability of success; subtracting the second linear/additive predictor from the first gives that logistic regression linear/additive predictor. That is, $logit(p) = \eta_1 - \eta_2$. This is illustated in one of the examples below.

The *extended* beta-binomial distribution of Prentice (1986) implemented by `extbetabinomial` is the preferred **VGAM** family function for BBD regression.

Author(s)

T. W. Yee

References

Moore, D. F. and Tsiatis, A. (1991). Robust estimation of the variance in moment methods for extra-binomial and extra-Poisson variation. *Biometrics*, **47**, 383–401.

Prentice, R. L. (1986). Binary regression using an extended beta-binomial distribution, with discussion of correlation induced by covariate measurement errors. *Journal of the American Statistical Association*, **81**, 321–327.

See Also

extbetabinomial, betabinomial, Betabinom, binomialff, betaff, dirmultinomial, lirat, simulate.vlm.

Examples

```
# Example 1
N <- 10; s1 <- exp(1); s2 <- exp(2)
y <- rbetabinom.ab(n = 100, size = N, shape1 = s1, shape2 = s2)
fit <- vglm(cbind(y, N-y) ~ 1, betabinomialff, trace = TRUE)
coef(fit, matrix = TRUE)
Coef(fit)
head(fit@misc$rho)  # The correlation parameter
head(cbind(depvar(fit), weights(fit, type = "prior")))


# Example 2
fit <- vglm(cbind(R, N-R) ~ 1, betabinomialff, data = lirat,
            trace = TRUE, subset = N > 1)
coef(fit, matrix = TRUE)
Coef(fit)
fit@misc$rho  # The correlation parameter
t(fitted(fit))
t(depvar(fit))
t(weights(fit, type = "prior"))
# A "loglink" link for the 2 shape params is a logistic regression:
all.equal(c(fitted(fit)),
          as.vector(logitlink(predict(fit)[, 1] -
                              predict(fit)[, 2], inverse = TRUE)))


# Example 3, which is more complicated
lirat <- transform(lirat, fgrp = factor(grp))
summary(lirat)  # Only 5 litters in group 3
fit2 <- vglm(cbind(R, N-R) ~ fgrp + hb, betabinomialff(zero = 2),
           data = lirat, trace = TRUE, subset = N > 1)
coef(fit2, matrix = TRUE)
coef(fit2, matrix = TRUE)[, 1] -
coef(fit2, matrix = TRUE)[, 2]  # logitlink(p)
## Not run:  with(lirat, plot(hb[N > 1], fit2@misc$rho,
   xlab = "Hemoglobin", ylab = "Estimated rho",
   pch = as.character(grp[N > 1]), col = grp[N > 1]))
## End(Not run)
```

```
## Not run:   # cf. Figure 3 of Moore and Tsiatis (1991)
with(lirat, plot(hb, R / N, pch = as.character(grp), col = grp,
   xlab = "Hemoglobin level", ylab = "Proportion Dead", las = 1,
   main = "Fitted values (lines)"))

smalldf <- with(lirat, lirat[N > 1, ])
for (gp in 1:4) {
  xx <- with(smalldf, hb[grp == gp])
  yy <- with(smalldf, fitted(fit2)[grp == gp])
  ooo <- order(xx)
  lines(xx[ooo], yy[ooo], col = gp, lwd = 2)
}
## End(Not run)
```

---

betaff *The Two-parameter Beta Distribution Family Function*

Description

Estimation of the mean and precision parameters of the beta distribution.

Usage

```
betaff(A = 0, B = 1, lmu = "logitlink", lphi = "loglink",
       imu = NULL, iphi = NULL,
       gprobs.y = ppoints(8), gphi  = exp(-3:5)/4, zero = NULL)
```

Arguments

`A, B`	Lower and upper limits of the distribution. The defaults correspond to the *standard beta distribution* where the response lies between 0 and 1.
`lmu, lphi`	Link function for the mean and precision parameters. The values A and B are extracted from the `min` and `max` arguments of `extlogitlink`. Consequently, only `extlogitlink` is allowed.
`imu, iphi`	Optional initial value for the mean and precision parameters respectively. A `NULL` value means a value is obtained in the `initialize` slot.
`gprobs.y, gphi, zero`	See `CommonVGAMffArguments` for more information.

Details

The two-parameter beta distribution can be written $f(y) =$

$$(y - A)^{\mu_1 \phi - 1} \times (B - y)^{(1-\mu_1)\phi - 1} / [beta(\mu_1 \phi, (1 - \mu_1)\phi) \times (B - A)^{\phi - 1}]$$

for $A < y < B$, and $beta(.,.)$ is the beta function (see `beta`). The parameter μ_1 satisfies $\mu_1 = (\mu - A)/(B - A)$ where μ is the mean of Y. That is, μ_1 is the mean of of a standard beta distribution:

$E(Y) = A + (B - A) \times \mu_1$, and these are the fitted values of the object. Also, ϕ is positive and $A < \mu < B$. Here, the limits A and B are *known*.

Another parameterization of the beta distribution involving the raw shape parameters is implemented in betaR.

For general A and B, the variance of Y is $(B - A)^2 \times \mu_1 \times (1 - \mu_1)/(1 + \phi)$. Then ϕ can be interpreted as a *precision* parameter in the sense that, for fixed μ, the larger the value of ϕ, the smaller the variance of Y. Also, $\mu_1 = shape1/(shape1 + shape2)$ and $\phi = shape1 + shape2$. Fisher scoring is implemented.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Note

The response must have values in the interval (A, B). The user currently needs to manually choose lmu to match the input of arguments A and B, e.g., with extlogitlink; see the example below.

Author(s)

Thomas W. Yee

References

Ferrari, S. L. P. and Francisco C.-N. (2004). Beta regression for modelling rates and proportions. *Journal of Applied Statistics*, **31**, 799–815.

See Also

betaR,

Beta, dzoabeta, genbetaII, betaII, betabinomialff, betageometric, betaprime, rbetageom, rbetanorm, kumar, extlogitlink, simulate.vlm.

Examples

```
bdata <- data.frame(y = rbeta(nn <- 1000, shape1 = exp(0),
                              shape2 = exp(1)))
fit1 <- vglm(y ~ 1, betaff, data = bdata, trace = TRUE)
coef(fit1, matrix = TRUE)
Coef(fit1)  # Useful for intercept-only models

# General A and B, and with a covariate
bdata <- transform(bdata, x2 = runif(nn))
bdata <- transform(bdata, mu = logitlink(0.5 - x2, inverse = TRUE),
                          prec = exp(3.0 + x2))  # prec == phi
bdata <- transform(bdata, shape2 = prec * (1 - mu),
                          shape1 = mu * prec)
bdata <- transform(bdata,
                   y = rbeta(nn, shape1 = shape1, shape2 = shape2))
```

```
bdata <- transform(bdata, Y = 5 + 8 * y)  # From 5--13, not 0--1
fit <- vglm(Y ~ x2, data = bdata, trace = TRUE,
   betaff(A = 5, B = 13, lmu = extlogitlink(min = 5, max = 13)))
coef(fit, matrix = TRUE)
```

Betageom *The Beta-Geometric Distribution*

Description

Density, distribution function, and random generation for the beta-geometric distribution.

Usage

```
dbetageom(x, shape1, shape2, log = FALSE)
pbetageom(q, shape1, shape2, log.p = FALSE)
rbetageom(n, shape1, shape2)
```

Arguments

x, q	vector of quantiles.
n	number of observations. Same as runif.
shape1, shape2	the two (positive) shape parameters of the standard beta distribution. They are called a and b in beta respectively.
log, log.p	Logical. If TRUE then all probabilities p are given as log(p).

Details

The beta-geometric distribution is a geometric distribution whose probability of success is not a constant but it is generated from a beta distribution with parameters shape1 and shape2. Note that the mean of this beta distribution is shape1/(shape1+shape2), which therefore is the mean of the probability of success.

Value

dbetageom gives the density, pbetageom gives the distribution function, and rbetageom generates random deviates.

Note

pbetageom can be particularly slow.

Author(s)

T. W. Yee

See Also

geometric, betaff, Beta.

Examples

```
## Not run:
shape1 <- 1; shape2 <- 2; y <- 0:30
proby <- dbetageom(y, shape1, shape2, log = FALSE)
plot(y, proby, type = "h", col = "blue", ylab = "P[Y=y]", main = paste0(
     "Y ~ Beta-geometric(shape1=", shape1,", shape2=", shape2, ")"))
sum(proby)

## End(Not run)
```

---

betageometric *Beta-geometric Distribution Family Function*

Description

Maximum likelihood estimation for the beta-geometric distribution.

Usage

```
betageometric(lprob = "logitlink", lshape = "loglink",
    iprob = NULL,    ishape = 0.1,
    moreSummation = c(2, 100), tolerance = 1.0e-10, zero = NULL)
```

Arguments

lprob, lshape	Parameter link functions applied to the parameters p and ϕ (called prob and shape below). The former lies in the unit interval and the latter is positive. See Links for more choices.
iprob, ishape	Numeric. Initial values for the two parameters. A NULL means a value is computed internally.
moreSummation	Integer, of length 2. When computing the expected information matrix a series summation from 0 to moreSummation[1]*max(y)+moreSummation[2] is made, in which the upper limit is an approximation to infinity. Here, y is the response.
tolerance	Positive numeric. When all terms are less than this then the series is deemed to have converged.
zero	An integer-valued vector specifying which linear/additive predictors are modelled as intercepts only. If used, the value must be from the set {1,2}. See CommonVGAMffArguments for more information.

Details

A random variable Y has a 2-parameter beta-geometric distribution if $P(Y = y) = p(1-p)^y$ for $y = 0, 1, 2, \ldots$ where p are generated from a standard beta distribution with shape parameters shape1 and shape2. The parameterization here is to focus on the parameters p and $\phi = 1/(shape1 + shape2)$, where ϕ is shape. The default link functions for these ensure that the appropriate range of the parameters is maintained. The mean of Y is $E(Y) = shape2/(shape1 - 1) = (1-p)/(p-\phi)$ if shape1 > 1, and if so, then this is returned as the fitted values.

The geometric distribution is a special case of the beta-geometric distribution with $\phi = 0$ (see geometric). However, fitting data from a geometric distribution may result in numerical problems because the estimate of $\log(\phi)$ will 'converge' to -Inf.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Note

The first iteration may be very slow; if practical, it is best for the weights argument of vglm etc. to be used rather than inputting a very long vector as the response, i.e., vglm(y ~ 1, ..., weights = wts) is to be preferred over vglm(rep(y, wts) ~ 1, ...). If convergence problems occur try inputting some values of argument ishape.

If an intercept-only model is fitted then the misc slot of the fitted object has list components shape1 and shape2.

Author(s)

T. W. Yee

References

Paul, S. R. (2005). Testing goodness of fit of the geometric distribution: an application to human fecundability data. *Journal of Modern Applied Statistical Methods*, **4**, 425–433.

See Also

geometric, betaff, rbetageom.

Examples

```
bdata <- data.frame(y = 0:11,
                    wts = c(227,123,72,42,21,31,11,14,6,4,7,28))
fitb <- vglm(y ~ 1, betageometric, bdata, weight = wts, trace = TRUE)
fitg <- vglm(y ~ 1,     geometric, bdata, weight = wts, trace = TRUE)
coef(fitb, matrix = TRUE)
Coef(fitb)
sqrt(diag(vcov(fitb, untransform = TRUE)))
fitb@misc$shape1
fitb@misc$shape2
```

```
# Very strong evidence of a beta-geometric:
pchisq(2 * (logLik(fitb) - logLik(fitg)), df = 1, lower.tail = FALSE)
```

---

betaII *Beta Distribution of the Second Kind*

Description

Maximum likelihood estimation of the 3-parameter beta II distribution.

Usage

```
betaII(lscale = "loglink", lshape2.p = "loglink",
       lshape3.q = "loglink", iscale = NULL, ishape2.p = NULL,
       ishape3.q = NULL, imethod = 1,
       gscale = exp(-5:5), gshape2.p = exp(-5:5),
       gshape3.q = seq(0.75, 4, by = 0.25),
       probs.y = c(0.25, 0.5, 0.75), zero = "shape")
```

Arguments

`lscale, lshape2.p, lshape3.q`
: Parameter link functions applied to the (positive) parameters `scale`, `p` and `q`. See `Links` for more choices.

`iscale, ishape2.p, ishape3.q, imethod, zero`
: See `CommonVGAMffArguments` for information.

`gscale, gshape2.p, gshape3.q`
: See `CommonVGAMffArguments` for information.

`probs.y`
: See `CommonVGAMffArguments` for information.

Details

The 3-parameter beta II is the 4-parameter *generalized* beta II distribution with shape parameter $a = 1$. It is also known as the Pearson VI distribution. Other distributions which are special cases of the 3-parameter beta II include the Lomax ($p = 1$) and inverse Lomax ($q = 1$). More details can be found in Kleiber and Kotz (2003).

The beta II distribution has density

$$f(y) = y^{p-1}/[b^p B(p, q)\{1 + y/b\}^{p+q}]$$

for $b > 0$, $p > 0$, $q > 0$, $y \geq 0$. Here, b is the scale parameter `scale`, and the others are shape parameters. The mean is

$$E(Y) = b\,\Gamma(p + 1)\,\Gamma(q - 1)/(\Gamma(p)\,\Gamma(q))$$

provided $q > 1$; these are returned as the fitted values. This family function handles multiple responses.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Note

See the notes in genbetaII.

Author(s)

T. W. Yee

References

Kleiber, C. and Kotz, S. (2003). *Statistical Size Distributions in Economics and Actuarial Sciences*, Hoboken, NJ, USA: Wiley-Interscience.

See Also

betaff, genbetaII, dagum, sinmad, fisk, inv.lomax, lomax, paralogistic, inv.paralogistic.

Examples

```
bdata <- data.frame(y = rsinmad(2000, shape1.a = 1,
         shape3.q = exp(2), scale = exp(1)))  # Not genuine data!
# fit <- vglm(y ~ 1, betaII, data = bdata, trace = TRUE)
fit <- vglm(y ~ 1, betaII(ishape2.p = 0.7, ishape3.q = 0.7),
            data = bdata, trace = TRUE)
coef(fit, matrix = TRUE)
Coef(fit)
summary(fit)
```

Betanorm *The Beta-Normal Distribution*

Description

Density, distribution function, quantile function and random generation for the univariate beta-normal distribution.

Usage

```
dbetanorm(x, shape1, shape2, mean = 0, sd = 1, log = FALSE)
pbetanorm(q, shape1, shape2, mean = 0, sd = 1,
          lower.tail = TRUE, log.p = FALSE)
qbetanorm(p, shape1, shape2, mean = 0, sd = 1,
          lower.tail = TRUE, log.p = FALSE)
rbetanorm(n, shape1, shape2, mean = 0, sd = 1)
```

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
n	number of observations. Same as runif.
shape1, shape2	the two (positive) shape parameters of the standard beta distribution. They are called a and b respectively in beta.
mean, sd	the mean and standard deviation of the univariate normal distribution (Normal).
log, log.p	Logical. If TRUE then all probabilities p are given as log(p).
lower.tail	Logical. If TRUE then the upper tail is returned, i.e., one minus the usual answer.

Details

The function betauninormal, the **VGAM** family function for estimating the parameters, has not yet been written.

Value

dbetanorm gives the density, pbetanorm gives the distribution function, qbetanorm gives the quantile function, and rbetanorm generates random deviates.

Author(s)

T. W. Yee

References

Gupta, A. K. and Nadarajah, S. (2004). *Handbook of Beta Distribution and Its Applications*, pp.146–152. New York: Marcel Dekker.

Examples

```
## Not run:
shape1 <- 0.1; shape2 <- 4; m <- 1
x <- seq(-10, 2, len = 501)
plot(x, dbetanorm(x, shape1, shape2, m = m), type = "l",
     ylim = 0:1, las = 1,
     ylab = paste0("betanorm(",shape1,", ",shape2,", m=",m, ", sd=1)"),
     main = "Blue is density, orange is the CDF",
     sub = "Gray lines are the 10,20,...,90 percentiles", col = "blue")
lines(x, pbetanorm(x, shape1, shape2, m = m), col = "orange")
abline(h = 0, col = "black")
probs <- seq(0.1, 0.9, by = 0.1)
Q <- qbetanorm(probs, shape1, shape2, m = m)
lines(Q, dbetanorm(Q, shape1, shape2, m = m),
      col = "gray50", lty = 2, type = "h")
lines(Q, pbetanorm(Q, shape1, shape2, m = m),
      col = "gray50", lty = 2, type = "h")
abline(h = probs, col = "gray50", lty = 2)
pbetanorm(Q, shape1, shape2, m = m) - probs  # Should be all 0
```

```
## End(Not run)
```

betaprime *The Beta-Prime Distribution*

Description

Estimation of the two shape parameters of the beta-prime distribution by maximum likelihood estimation.

Usage

```
betaprime(lshape = "loglink", ishape1 = 2, ishape2 = NULL,
          zero = NULL)
```

Arguments

`lshape` Parameter link function applied to the two (positive) shape parameters. See `Links` for more choices.

`ishape1, ishape2, zero` See `CommonVGAMffArguments` for more information.

Details

The beta-prime distribution is given by

$$f(y) = y^{shape1-1}(1+y)^{-shape1-shape2}/B(shape1, shape2)$$

for $y > 0$. The shape parameters are positive, and here, B is the beta function. The mean of Y is $shape1/(shape2-1)$ provided $shape2 > 1$; these are returned as the fitted values.

If Y has a $Beta(shape1, shape2)$ distribution then $Y/(1-Y)$ and $(1-Y)/Y$ have a $Betaprime(shape1, shape2)$ and $Betaprime(shape2, shape1)$ distribution respectively. Also, if Y_1 has a $gamma(shape1)$ distribution and Y_2 has a $gamma(shape2)$ distribution then Y_1/Y_2 has a $Betaprime(shape1, shape2)$ distribution.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, `rrvglm` and `vgam`.

Note

The response must have positive values only.

The beta-prime distribution is also known as the *beta distribution of the second kind* or the *inverted beta distribution*.

Author(s)

Thomas W. Yee

References

Johnson, N. L. and Kotz, S. and Balakrishnan, N. (1995). Chapter 25 of: *Continuous Univariate Distributions*, 2nd edition, Volume 2, New York: Wiley.

See Also

betaff, Beta.

Examples

```
nn <- 1000
bdata <- data.frame(shape1 = exp(1), shape2 = exp(3))
bdata <- transform(bdata, yb = rbeta(nn, shape1, shape2))
bdata <- transform(bdata, y1 = (1-yb) /    yb,
                          y2 =    yb  / (1-yb),
                          y3 = rgamma(nn, exp(3)) / rgamma(nn, exp(2)))

fit1 <- vglm(y1 ~ 1, betaprime, data = bdata, trace = TRUE)
coef(fit1, matrix = TRUE)

fit2 <- vglm(y2 ~ 1, betaprime, data = bdata, trace = TRUE)
coef(fit2, matrix = TRUE)

fit3 <- vglm(y3 ~ 1, betaprime, data = bdata, trace = TRUE)
coef(fit3, matrix = TRUE)

# Compare the fitted values
with(bdata, mean(y3))
head(fitted(fit3))
Coef(fit3)  # Useful for intercept-only models
```

---

betaR *The Two-parameter Beta Distribution Family Function*

Description

Estimation of the shape parameters of the two-parameter beta distribution.

Usage

```
betaR(lshape1 = "loglink", lshape2 = "loglink",
      i1 = NULL, i2 = NULL, trim = 0.05,
      A = 0, B = 1, parallel = FALSE, zero = NULL)
```

Arguments

`lshape1, lshape2, i1, i2`
: Details at `CommonVGAMffArguments`. See `Links` for more choices.

`trim`
: An argument which is fed into `mean()`; it is the fraction (0 to 0.5) of observations to be trimmed from each end of the response `y` before the mean is computed. This is used when computing initial values, and guards against outliers.

`A, B`
: Lower and upper limits of the distribution. The defaults correspond to the *standard beta distribution* where the response lies between 0 and 1.

`parallel, zero`
: See `CommonVGAMffArguments` for more information.

Details

The two-parameter beta distribution is given by $f(y) =$

$$(y - A)^{shape1-1} \times (B - y)^{shape2-1} / [Beta(shape1, shape2) \times (B - A)^{shape1+shape2-1}]$$

for $A < y < B$, and $Beta(.,.)$ is the beta function (see `beta`). The shape parameters are positive, and here, the limits A and B are known. The mean of Y is $E(Y) = A + (B - A) \times shape1/(shape1 + shape2)$, and these are the fitted values of the object.

For the standard beta distribution the variance of Y is $shape1 \times shape2/[(1+shape1+shape2) \times (shape1 + shape2)^2]$. If $\sigma^2 = 1/(1 + shape1 + shape2)$ then the variance of Y can be written $\sigma^2 \mu(1 - \mu)$ where $\mu = shape1/(shape1 + shape2)$ is the mean of Y.

Another parameterization of the beta distribution involving the mean and a precision parameter is implemented in `betaff`.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, `rrvglm` and `vgam`.

Note

The response must have values in the interval (A, B). **VGAM** 0.7-4 and prior called this function `betaff`.

Author(s)

Thomas W. Yee

References

Johnson, N. L. and Kotz, S. and Balakrishnan, N. (1995). Chapter 25 of: *Continuous Univariate Distributions*, 2nd edition, Volume 2, New York: Wiley.

Gupta, A. K. and Nadarajah, S. (2004). *Handbook of Beta Distribution and Its Applications*, New York: Marcel Dekker.

See Also

betaff,

Beta, genbetaII, betaII, betabinomialff, betageometric, betaprime, rbetageom, rbetanorm, kumar, simulate.vlm.

Examples

```
bdata <- data.frame(y = rbeta(1000, shape1 = exp(0), shape2 = exp(1)))
fit <- vglm(y ~ 1, betaR(lshape1 = "identitylink",
            lshape2 = "identitylink"), bdata, trace = TRUE, crit = "coef")
fit <- vglm(y ~ 1, betaR, data = bdata, trace = TRUE, crit = "coef")
coef(fit, matrix = TRUE)
Coef(fit)  # Useful for intercept-only models

bdata <- transform(bdata, Y = 5 + 8 * y)  # From 5 to 13, not 0 to 1
fit <- vglm(Y ~ 1, betaR(A = 5, B = 13), data = bdata, trace = TRUE)
Coef(fit)
c(meanY = with(bdata, mean(Y)), head(fitted(fit),2))
```

Biamhcop *Ali-Mikhail-Haq Bivariate Distribution*

Description

Density, distribution function, and random generation for the (one parameter) bivariate Ali-Mikhail-Haq distribution.

Usage

```
dbiamhcop(x1, x2, apar, log = FALSE)
pbiamhcop(q1, q2, apar)
rbiamhcop(n, apar)
```

Arguments

x1, x2, q1, q2	vector of quantiles.
n	number of observations. Same as runif
apar	the association parameter.
log	Logical. If TRUE then the logarithm is returned.

Details

See biamhcop, the **VGAM** family functions for estimating the parameter by maximum likelihood estimation, for the formula of the cumulative distribution function and other details.

Value

dbiamhcop gives the density, pbiamhcop gives the distribution function, and rbiamhcop generates random deviates (a two-column matrix).

Author(s)

T. W. Yee and C. S. Chee

See Also

biamhcop.

Examples

```
 x <- seq(0, 1, len = (N <- 101)); apar <- 0.7
ox <- expand.grid(x, x)
zedd <- dbiamhcop(ox[, 1], ox[, 2], apar = apar)
## Not run:
contour(x, x, matrix(zedd, N, N), col = "blue")
zedd <- pbiamhcop(ox[, 1], ox[, 2], apar = apar)
contour(x, x, matrix(zedd, N, N), col = "blue")

plot(r <- rbiamhcop(n = 1000, apar = apar), col = "blue")
par(mfrow = c(1, 2))
hist(r[, 1])  # Should be uniform
hist(r[, 2])  # Should be uniform

## End(Not run)
```

---

biamhcop *Ali-Mikhail-Haq Distribution Family Function*

Description

Estimate the association parameter of Ali-Mikhail-Haq's bivariate distribution by maximum likelihood estimation.

Usage

```
biamhcop(lapar = "rhobitlink", iapar = NULL, imethod = 1,
         nsimEIM = 250)
```

Arguments

lapar
: Link function applied to the association parameter α, which is real and $-1 < \alpha < 1$. See Links for more choices.

iapar
: Numeric. Optional initial value for α. By default, an initial value is chosen internally. If a convergence failure occurs try assigning a different value. Assigning a value will override the argument imethod.

imethod An integer with value 1 or 2 which specifies the initialization method. If failure to converge occurs try the other value, or else specify a value for iapar.

nsimEIM See CommonVGAMffArguments for more information.

Details

The cumulative distribution function is

$$P(Y_1 \leq y_1, Y_2 \leq y_2) = y_1 y_2 / (1 - \alpha(1 - y_1)(1 - y_2))$$

for $-1 < \alpha < 1$. The support of the function is the unit square. The marginal distributions are the standard uniform distributions. When $\alpha = 0$ the random variables are independent. This is an Archimedean copula.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm and vgam.

Note

The response must be a two-column matrix. Currently, the fitted value is a matrix with two columns and values equal to 0.5. This is because each marginal distribution corresponds to a standard uniform distribution.

Author(s)

T. W. Yee and C. S. Chee

References

Balakrishnan, N. and Lai, C.-D. (2009). *Continuous Bivariate Distributions*, 2nd ed. New York: Springer.

See Also

rbiamhcop, bifgmcop, bigumbelIexp, rbilogis, simulate.vlm.

Examples

```
ymat <- rbiamhcop(1000, apar = rhobitlink(2, inverse = TRUE))
fit <- vglm(ymat ~ 1, biamhcop, trace = TRUE)
coef(fit, matrix = TRUE)
Coef(fit)
```

Biclaytoncop *Clayton Copula (Bivariate) Distribution*

Description

Density and random generation for the (one parameter) bivariate Clayton copula distribution.

Usage

```
dbiclaytoncop(x1, x2, apar = 0, log = FALSE)
rbiclaytoncop(n, apar = 0)
```

Arguments

- `x1, x2` vector of quantiles. The `x1` and `x2` should both be in the interval $(0, 1)$.
- `n` number of observations. Same as `rnorm`.
- `apar` the association parameter. Should be in the interval $[0, \infty)$. The default corresponds to independence.
- `log` Logical. If `TRUE` then the logarithm is returned.

Details

See `biclaytoncop`, the **VGAM** family functions for estimating the parameter by maximum likelihood estimation, for the formula of the cumulative distribution function and other details.

Value

`dbiclaytoncop` gives the density at point (`x1`,`x2`), `rbiclaytoncop` generates random deviates (a two-column matrix).

Note

`dbiclaytoncop()` does not yet handle `x1 = 0` and/or `x2 = 0`.

Author(s)

R. Feyter and T. W. Yee

References

Clayton, D. (1982). A model for association in bivariate survival data. *Journal of the Royal Statistical Society, Series B, Methodological*, **44**, 414–422.

See Also

`biclaytoncop`, `binormalcop`, `binormal`.

Examples

```
## Not run:  edge <- 0.01  # A small positive value
N <- 101; x <- seq(edge, 1.0 - edge, len = N); Rho <- 0.7
ox <- expand.grid(x, x)
zedd <- dbiclaytoncop(ox[, 1], ox[, 2], apar = Rho, log = TRUE)
par(mfrow = c(1, 2))
contour(x, x, matrix(zedd, N, N), col = 4, labcex = 1.5, las = 1)
plot(rbiclaytoncop(1000, 2), col = 4, las = 1)
## End(Not run)
```

`biclaytoncop` *Clayton Copula (Bivariate) Family Function*

Description

Estimate the correlation parameter of the (bivariate) Clayton copula distribution by maximum likelihood estimation.

Usage

```
biclaytoncop(lapar = "loglink", iapar = NULL, imethod = 1,
             parallel = FALSE, zero = NULL)
```

Arguments

`lapar, iapar, imethod`
Details at `CommonVGAMffArguments`. See `Links` for more link function choices.

`parallel, zero` Details at `CommonVGAMffArguments`. If `parallel = TRUE` then the constraint is also applied to the intercept.

Details

The cumulative distribution function is

$$P(u_1, u_2; \alpha) = (u_1^{-\alpha} + u_2^{-\alpha} - 1)^{-1/\alpha}$$

for $0 \leq \alpha$. Here, α is the association parameter. The support of the function is the interior of the unit square; however, values of 0 and/or 1 are not allowed (currently). The marginal distributions are the standard uniform distributions. When $\alpha = 0$ the random variables are independent.

This **VGAM** family function can handle multiple responses, for example, a six-column matrix where the first 2 columns is the first out of three responses, the next 2 columns being the next response, etc.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Note

The response matrix must have a multiple of two-columns. Currently, the fitted value is a matrix with the same number of columns and values equal to 0.5. This is because each marginal distribution corresponds to a standard uniform distribution.

This **VGAM** family function is fragile; each response must be in the interior of the unit square.

Author(s)

R. Feyter and T. W. Yee

References

Clayton, D. (1982). A model for association in bivariate survival data. *Journal of the Royal Statistical Society, Series B, Methodological*, **44**, 414–422.

Schepsmeier, U. and Stober, J. (2014). Derivatives and Fisher information of bivariate copulas. *Statistical Papers* **55**, 525–542.

See Also

rbiclaytoncop, dbiclaytoncop, kendall.tau.

Examples

```
ymat <- rbiclaytoncop(n = (nn <- 1000), apar = exp(2))
bdata <- data.frame(y1 = ymat[, 1], y2 = ymat[, 2],
                    y3 = ymat[, 1], y4 = ymat[, 2], x2 = runif(nn))
summary(bdata)
## Not run:  plot(ymat, col = "blue")
fit1 <-
  vglm(cbind(y1, y2, y3, y4) ~ 1,  # 2 responses, e.g., (y1,y2) is the 1st
       biclaytoncop, data = bdata,
       trace = TRUE, crit = "coef")  # Sometimes a good idea
coef(fit1, matrix = TRUE)
Coef(fit1)
head(fitted(fit1))
summary(fit1)

# Another example; apar is a function of x2
bdata <- transform(bdata, apar = exp(-0.5 + x2))
ymat <- rbiclaytoncop(n = nn, apar = with(bdata, apar))
bdata <- transform(bdata, y5 = ymat[, 1], y6 = ymat[, 2])
fit2 <- vgam(cbind(y5, y6) ~ s(x2), data = bdata,
             biclaytoncop(lapar = "loglink"), trace = TRUE)
## Not run: plot(fit2, lcol = "blue", scol = "orange", se = TRUE)
```

BICvlm *Bayesian Information Criterion*

Description

Calculates the Bayesian information criterion (BIC) for a fitted model object for which a log-likelihood value has been obtained.

Usage

```
BICvlm(object, ..., k = log(nobs(object)))
```

Arguments

`object, ...`	Same as `AICvlm`.
`k`	Numeric, the penalty per parameter to be used; the default is `log(n)` where `n` is the number of observations).

Details

The so-called BIC or SBC (Schwarz's Bayesian criterion) can be computed by calling `AICvlm` with a different `k` argument. See `AICvlm` for information and caveats.

Value

Returns a numeric value with the corresponding BIC, or . . . , depending on `k`.

Warning

Like `AICvlm`, this code has not been double-checked. The general applicability of `BIC` for the VGLM/VGAM classes has not been developed fully. In particular, `BIC` should not be run on some **VGAM** family functions because of violation of certain regularity conditions, etc.

Many **VGAM** family functions such as `cumulative` can have the number of observations absorbed into the prior weights argument (e.g., `weights` in `vglm`), either before or after fitting. Almost all **VGAM** family functions can have the number of observations defined by the `weights` argument, e.g., as an observed frequency. `BIC` simply uses the number of rows of the model matrix, say, as defining `n`, hence the user must be very careful of this possible error. Use at your own risk!!

Note

BIC, AIC and other ICs can have have many additive constants added to them. The important thing are the differences since the minimum value corresponds to the best model.

BIC has not been defined for QRR-VGLMs yet.

Author(s)

T. W. Yee.

See Also

AICvlm, VGLMs are described in vglm-class; VGAMs are described in vgam-class; RR-VGLMs are described in rrvglm-class; BIC, AIC.

Examples

```
pneumo <- transform(pneumo, let = log(exposure.time))
(fit1 <- vglm(cbind(normal, mild, severe) ~ let,
              cumulative(parallel = TRUE, reverse = TRUE), data = pneumo))
coef(fit1, matrix = TRUE)
BIC(fit1)
(fit2 <- vglm(cbind(normal, mild, severe) ~ let,
              cumulative(parallel = FALSE, reverse = TRUE), data = pneumo))
coef(fit2, matrix = TRUE)
BIC(fit2)
```

Bifgmcop *Farlie-Gumbel-Morgenstern's Bivariate Distribution*

Description

Density, distribution function, and random generation for the (one parameter) bivariate Farlie-Gumbel-Morgenstern's distribution.

Usage

```
dbifgmcop(x1, x2, apar, log = FALSE)
pbifgmcop(q1, q2, apar)
rbifgmcop(n, apar)
```

Arguments

x1, x2, q1, q2	vector of quantiles.
n	number of observations. Same as in runif.
apar	the association parameter.
log	Logical. If TRUE then the logarithm is returned.

Details

See bifgmcop, the **VGAM** family functions for estimating the parameter by maximum likelihood estimation, for the formula of the cumulative distribution function and other details.

Value

dbifgmcop gives the density, pbifgmcop gives the distribution function, and rbifgmcop generates random deviates (a two-column matrix).

Author(s)

T. W. Yee

See Also

bifgmcop.

Examples

```
## Not run:  N <- 101; x <- seq(0.0, 1.0, len = N); apar <- 0.7
ox <- expand.grid(x, x)
zedd <- dbifgmcop(ox[, 1], ox[, 2], apar = apar)
contour(x, x, matrix(zedd, N, N), col = "blue")
zedd <- pbifgmcop(ox[, 1], ox[, 2], apar = apar)
contour(x, x, matrix(zedd, N, N), col = "blue")

plot(r <- rbifgmcop(n = 3000, apar = apar), col = "blue")
par(mfrow = c(1, 2))
hist(r[, 1])  # Should be uniform
hist(r[, 2])  # Should be uniform

## End(Not run)
```

---

bifgmcop — *Farlie-Gumbel-Morgenstern's Bivariate Distribution Family Function*

Description

Estimate the association parameter of Farlie-Gumbel-Morgenstern's bivariate distribution by maximum likelihood estimation.

Usage

```
bifgmcop(lapar = "rhobitlink", iapar = NULL, imethod = 1)
```

Arguments

`lapar, iapar, imethod`
: Details at CommonVGAMffArguments. See Links for more link function choices.

Details

The cumulative distribution function is

$$P(Y_1 \leq y_1, Y_2 \leq y_2) = y_1 y_2 (1 + \alpha(1 - y_1)(1 - y_2))$$

for $-1 < \alpha < 1$. The support of the function is the unit square. The marginal distributions are the standard uniform distributions. When $\alpha = 0$ the random variables are independent.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm and vgam.

Note

The response must be a two-column matrix. Currently, the fitted value is a matrix with two columns and values equal to 0.5. This is because each marginal distribution corresponds to a standard uniform distribution.

Author(s)

T. W. Yee

References

Castillo, E., Hadi, A. S., Balakrishnan, N. and Sarabia, J. S. (2005). *Extreme Value and Related Models with Applications in Engineering and Science*, Hoboken, NJ, USA: Wiley-Interscience.

Smith, M. D. (2007). Invariance theorems for Fisher information. *Communications in Statistics—Theory and Methods*, **36**(12), 2213–2222.

See Also

rbifgmcop, bifrankcop, bifgmexp, simulate.vlm.

Examples

```
ymat <- rbifgmcop(1000, apar = rhobitlink(3, inverse = TRUE))
## Not run: plot(ymat, col = "blue")
fit <- vglm(ymat ~ 1, fam = bifgmcop, trace = TRUE)
coef(fit, matrix = TRUE)
Coef(fit)
head(fitted(fit))
```

---

bifgmexp *Bivariate Farlie-Gumbel-Morgenstern Exponential Distribution Family Function*

Description

Estimate the association parameter of FGM bivariate exponential distribution by maximum likelihood estimation.

Usage

```
bifgmexp(lapar = "rhobitlink", iapar = NULL, tola0 = 0.01,
         imethod = 1)
```

Arguments

- `lapar` Link function for the association parameter α, which lies between -1 and 1. See `Links` for more choices and other information.
- `iapar` Numeric. Optional initial value for α. By default, an initial value is chosen internally. If a convergence failure occurs try assigning a different value. Assigning a value will override the argument `imethod`.
- `tola0` Positive numeric. If the estimate of α has an absolute value less than this then it is replaced by this value. This is an attempt to fix a numerical problem when the estimate is too close to zero.
- `imethod` An integer with value `1` or `2` which specifies the initialization method. If failure to converge occurs try the other value, or else specify a value for `ia`.

Details

The cumulative distribution function is

$$P(Y_1 \leq y_1, Y_2 \leq y_2) = e^{-y_1-y_2}(1+\alpha[1-e^{-y_1}][1-e^{-y_2}]) + 1 - e^{-y_1} - e^{-y_2}$$

for α between -1 and 1. The support of the function is for $y_1 > 0$ and $y_2 > 0$. The marginal distributions are an exponential distribution with unit mean. When $\alpha = 0$ then the random variables are independent, and this causes some problems in the estimation process since the distribution no longer depends on the parameter.

A variant of Newton-Raphson is used, which only seems to work for an intercept model. It is a very good idea to set `trace = TRUE`.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Note

The response must be a two-column matrix. Currently, the fitted value is a matrix with two columns and values equal to 1. This is because each marginal distribution corresponds to a exponential distribution with unit mean.

This **VGAM** family function should be used with caution.

Author(s)

T. W. Yee

References

Castillo, E., Hadi, A. S., Balakrishnan, N. and Sarabia, J. S. (2005). *Extreme Value and Related Models with Applications in Engineering and Science*, Hoboken, NJ, USA: Wiley-Interscience.

See Also

`bifgmcop`, `bigumbelIexp`.

Examples

```
N <- 1000; mdata <- data.frame(y1 = rexp(N), y2 = rexp(N))
## Not run: plot(ymat)
fit <- vglm(cbind(y1, y2) ~ 1, bifgmexp, data = mdata, trace = TRUE)
fit <- vglm(cbind(y1, y2) ~ 1, bifgmexp, data = mdata, # May fail
            trace = TRUE, crit = "coef")
coef(fit, matrix = TRUE)
Coef(fit)
head(fitted(fit))
```

---

bifrankcop *Frank's Bivariate Distribution Family Function*

Description

Estimate the association parameter of Frank's bivariate distribution by maximum likelihood estimation.

Usage

```
bifrankcop(lapar = "loglink", iapar = 2, nsimEIM = 250)
```

Arguments

`lapar`	Link function applied to the (positive) association parameter α. See `Links` for more choices.
`iapar`	Numeric. Initial value for α. If a convergence failure occurs try assigning a different value.
`nsimEIM`	See `CommonVGAMffArguments`.

Details

The cumulative distribution function is

$$P(Y_1 \leq y_1, Y_2 \leq y_2) = H_\alpha(y_1, y_2) = \log_\alpha[1 + (\alpha^{y_1} - 1)(\alpha^{y_2} - 1)/(\alpha - 1)]$$

for $\alpha \neq 1$. Note the logarithm here is to base α. The support of the function is the unit square.

When $0 < \alpha < 1$ the probability density function $h_\alpha(y_1, y_2)$ is symmetric with respect to the lines $y_2 = y_1$ and $y_2 = 1 - y_1$. When $\alpha > 1$ then $h_\alpha(y_1, y_2) = h_{1/\alpha}(1 - y_1, y_2)$.

$\alpha = 1$ then $H(y_1, y_2) = y_1 y_2$, i.e., uniform on the unit square. As α approaches 0 then $H(y_1, y_2) = \min(y_1, y_2)$. As α approaches infinity then $H(y_1, y_2) = \max(0, y_1 + y_2 - 1)$.

The default is to use Fisher scoring implemented using `rbifrankcop`. For intercept-only models an alternative is to set `nsimEIM=NULL` so that a variant of Newton-Raphson is used.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Note

The response must be a two-column matrix. Currently, the fitted value is a matrix with two columns and values equal to a half. This is because the marginal distributions correspond to a standard uniform distribution.

Author(s)

T. W. Yee

References

Genest, C. (1987). Frank's family of bivariate distributions. *Biometrika*, **74**, 549–555.

See Also

`rbifrankcop`, `bifgmcop`, `simulate.vlm`.

Examples

```
## Not run:
ymat <- rbifrankcop(n = 2000, apar = exp(4))
plot(ymat, col = "blue")
fit <- vglm(ymat ~ 1, fam = bifrankcop, trace = TRUE)
coef(fit, matrix = TRUE)
Coef(fit)
vcov(fit)
head(fitted(fit))
summary(fit)

## End(Not run)
```

`bigumbelIexp` *Gumbel's Type I Bivariate Distribution Family Function*

Description

Estimate the association parameter of Gumbel's Type I bivariate distribution by maximum likelihood estimation.

Usage

```
bigumbelIexp(lapar = "identitylink", iapar = NULL, imethod = 1)
```

Arguments

lapar	Link function applied to the association parameter α. See Links for more choices.
iapar	Numeric. Optional initial value for α. By default, an initial value is chosen internally. If a convergence failure occurs try assigning a different value. Assigning a value will override the argument imethod.
imethod	An integer with value 1 or 2 which specifies the initialization method. If failure to converge occurs try the other value, or else specify a value for ia.

Details

The cumulative distribution function is

$$P(Y_1 \leq y_1, Y_2 \leq y_2) = e^{-y_1 - y_2 + \alpha y_1 y_2} + 1 - e^{-y_1} - e^{-y_2}$$

for real α. The support of the function is for $y_1 > 0$ and $y_2 > 0$. The marginal distributions are an exponential distribution with unit mean.

A variant of Newton-Raphson is used, which only seems to work for an intercept model. It is a very good idea to set `trace=TRUE`.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm and vgam.

Note

The response must be a two-column matrix. Currently, the fitted value is a matrix with two columns and values equal to 1. This is because each marginal distribution corresponds to a exponential distribution with unit mean.

This **VGAM** family function should be used with caution.

Author(s)

T. W. Yee

References

Gumbel, E. J. (1960). Bivariate Exponential Distributions. *Journal of the American Statistical Association*, **55**, 698–707.

See Also

bifgmexp.

Examples

```
nn <- 1000
gdata <- data.frame(y1 = rexp(nn), y2 = rexp(nn))
## Not run:  with(gdata, plot(cbind(y1, y2)))
fit <- vglm(cbind(y1, y2) ~ 1, bigumbelIexp, gdata, trace = TRUE)
coef(fit, matrix = TRUE)
Coef(fit)
head(fitted(fit))
```

`bilogis` *Bivariate Logistic Distribution*

Description

Density, distribution function, quantile function and random generation for the 4-parameter bivariate logistic distribution.

Usage

```
dbilogis(x1, x2, loc1 = 0, scale1 = 1, loc2 = 0, scale2 = 1,
         log = FALSE)
pbilogis(q1, q2, loc1 = 0, scale1 = 1, loc2 = 0, scale2 = 1)
rbilogis(n, loc1 = 0, scale1 = 1, loc2 = 0, scale2 = 1)
```

Arguments

`x1, x2, q1, q2`	vector of quantiles.
`n`	number of observations. Same as `rlogis`.
`loc1, loc2`	the location parameters l_1 and l_2.
`scale1, scale2`	the scale parameters s_1 and s_2.
`log`	Logical. If `log = TRUE` then the logarithm of the density is returned.

Details

See `bilogis`, the **VGAM** family function for estimating the four parameters by maximum likelihood estimation, for the formula of the cumulative distribution function and other details.

Value

`dbilogis` gives the density, `pbilogis` gives the distribution function, and `rbilogis` generates random deviates (a two-column matrix).

Note

Gumbel (1961) proposed two bivariate logistic distributions with logistic distribution marginals, which he called Type I and Type II. The Type I is this one. The Type II belongs to the Morgenstern type. The `biamhcop` distribution has, as a special case, this distribution, which is when the random variables are independent.

Author(s)

T. W. Yee

References

Gumbel, E. J. (1961). Bivariate logistic distributions. *Journal of the American Statistical Association*, **56**, 335–349.

See Also

bilogistic, biamhcop.

Examples

```
## Not run:  par(mfrow = c(1, 3))
ymat <- rbilogis(n = 2000, loc1 = 5, loc2 = 7, scale2 = exp(1))
myxlim <- c(-2, 15); myylim <- c(-10, 30)
plot(ymat, xlim = myxlim, ylim = myylim)

N <- 100
x1 <- seq(myxlim[1], myxlim[2], len = N)
x2 <- seq(myylim[1], myylim[2], len = N)
ox <- expand.grid(x1, x2)
z <- dbilogis(ox[,1], ox[,2], loc1 = 5, loc2 = 7, scale2 = exp(1))
contour(x1, x2, matrix(z, N, N), main = "density")
z <- pbilogis(ox[,1], ox[,2], loc1 = 5, loc2 = 7, scale2 = exp(1))
contour(x1, x2, matrix(z, N, N), main = "cdf")
## End(Not run)
```

---

bilogistic *Bivariate Logistic Distribution Family Function*

Description

Estimates the four parameters of the bivariate logistic distribution by maximum likelihood estimation.

Usage

```
bilogistic(llocation = "identitylink", lscale = "loglink",
           iloc1 = NULL, iscale1 = NULL, iloc2 = NULL, iscale2 =
           NULL, imethod = 1, nsimEIM = 250, zero = NULL)
```

Arguments

`llocation` Link function applied to both location parameters l_1 and l_2. See `Links` for more choices.

`lscale` Parameter link function applied to both (positive) scale parameters s_1 and s_2. See `Links` for more choices.

`iloc1, iloc2` Initial values for the location parameters. By default, initial values are chosen internally using `imethod`. Assigning values here will override the argument `imethod`.

`iscale1, iscale2` Initial values for the scale parameters. By default, initial values are chosen internally using `imethod`. Assigning values here will override the argument `imethod`.

`imethod` An integer with value 1 or 2 which specifies the initialization method. If failure to converge occurs try the other value.

`nsimEIM, zero` See `CommonVGAMffArguments` for details.

Details

The four-parameter bivariate logistic distribution has a density that can be written as

$$f(y_1, y_2; l_1, s_1, l_2, s_2) = 2 \frac{\exp[-(y_1 - l_1)/s_1 - (y_2 - l_2)/s_2]}{s_1 s_2 \left(1 + \exp[-(y_1 - l_1)/s_1] + \exp[-(y_2 - l_2)/s_2]\right)^3}$$

where $s_1 > 0$ and $s_2 > 0$ are the scale parameters, and l_1 and l_2 are the location parameters. Each of the two responses are unbounded, i.e., $-\infty < y_j < \infty$. The mean of Y_1 is l_1 etc. The fitted values are returned in a 2-column matrix. The cumulative distribution function is

$$F(y_1, y_2; l_1, s_1, l_2, s_2) = \left(1 + \exp[-(y_1 - l_1)/s_1] + \exp[-(y_2 - l_2)/s_2]\right)^{-1}$$

The marginal distribution of Y_1 is

$$P(Y_1 \le y_1) = F(y_1; l_1, s_1) = \left(1 + \exp[-(y_1 - l_1)/s_1]\right)^{-1}.$$

By default, $\eta_1 = l_1$, $\eta_2 = \log(s_1)$, $\eta_3 = l_2$, $\eta_4 = \log(s_2)$ are the linear/additive predictors.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, `rrvglm` and `vgam`.

Author(s)

T. W. Yee

References

Gumbel, E. J. (1961). Bivariate logistic distributions. *Journal of the American Statistical Association*, **56**, 335–349.

Castillo, E., Hadi, A. S., Balakrishnan, N. and Sarabia, J. S. (2005). *Extreme Value and Related Models with Applications in Engineering and Science*, Hoboken, NJ, USA: Wiley-Interscience.

See Also

logistic, rbilogis.

Examples

```
## Not run:
ymat <- rbilogis(n <- 50, loc1 = 5, loc2 = 7, scale2 = exp(1))
plot(ymat)
bfit <- vglm(ymat ~ 1, family = bilogistic, trace = TRUE)
coef(bfit, matrix = TRUE)
Coef(bfit)
head(fitted(bfit))
vcov(bfit)
head(weights(bfit, type = "work"))
summary(bfit)

## End(Not run)
```

Binom2.or *Bivariate Odds Ratio Model*

Description

Density and random generation for a bivariate binary regression model using an odds ratio as the measure of dependency.

Usage

```
rbinom2.or(n, mu1,
    mu2 = if (exchangeable) mu1 else
    stop("argument 'mu2' not specified"),
    oratio = 1, exchangeable = FALSE, tol = 0.001,
    twoCols = TRUE, colnames = if (twoCols) c("y1","y2") else
    c("00", "01", "10", "11"),
    ErrorCheck = TRUE)
dbinom2.or(mu1, mu2 = if (exchangeable) mu1 else
    stop("'mu2' not specified"),
    oratio = 1, exchangeable = FALSE, tol = 0.001,
    colnames = c("00", "01", "10", "11"), ErrorCheck = TRUE)
```

Arguments

n	number of observations. Same as in runif. The arguments mu1, mu2, oratio are recycled to this value.
mu1, mu2	The marginal probabilities. Only mu1 is needed if exchangeable = TRUE. Values should be between 0 and 1.

oratio	Odds ratio. Must be numeric and positive. The default value of unity means the responses are statistically independent.
exchangeable	Logical. If TRUE, the two marginal probabilities are constrained to be equal.
twoCols	Logical. If TRUE, then a $n \times 2$ matrix of 1s and 0s is returned. If FALSE, then a $n \times 4$ matrix of 1s and 0s is returned.
colnames	The dimnames argument of matrix is assigned list(NULL, colnames).
tol	Tolerance for testing independence. Should be some small positive numerical value.
ErrorCheck	Logical. Do some error checking of the input parameters?

Details

The function rbinom2.or generates data coming from a bivariate binary response model. The data might be fitted with the **VGAM** family function binom2.or.

The function dbinom2.or does not really compute the density (because that does not make sense here) but rather returns the four joint probabilities.

Value

The function rbinom2.or returns either a 2 or 4 column matrix of 1s and 0s, depending on the argument twoCols.

The function dbinom2.or returns a 4 column matrix of joint probabilities; each row adds up to unity.

Author(s)

T. W. Yee

See Also

binom2.or.

Examples

```
nn <- 1000  # Example 1
ymat <- rbinom2.or(nn, mu1 = logitlink(1, inv = TRUE),
                   oratio = exp(2), exch = TRUE)
(mytab <- table(ymat[, 1], ymat[, 2], dnn = c("Y1", "Y2")))
(myor <- mytab["0","0"] * mytab["1","1"] / (mytab["1","0"] *
         mytab["0","1"]))
fit <- vglm(ymat ~ 1, binom2.or(exch = TRUE))
coef(fit, matrix = TRUE)

bdata <- data.frame(x2 = sort(runif(nn)))  # Example 2
bdata <- transform(bdata,
           mu1 = logitlink(-2 + 4 * x2, inverse = TRUE),
           mu2 = logitlink(-1 + 3 * x2, inverse = TRUE))
dmat <- with(bdata, dbinom2.or(mu1 = mu1, mu2 = mu2,
                               oratio = exp(2)))
```

```
ymat <- with(bdata, rbinom2.or(n = nn, mu1 = mu1, mu2 = mu2,
                               oratio = exp(2)))
fit2 <- vglm(ymat ~ x2, binom2.or, data = bdata)
coef(fit2, matrix = TRUE)
## Not run:
matplot(with(bdata, x2), dmat, lty = 1:4, col = 1:4,
        main = "Joint probabilities", ylim = 0:1, type = "l",
        ylab = "Probabilities", xlab = "x2", las = 1)
legend("top", lty = 1:4, col = 1:4,
       legend = c("1 = (y1=0, y2=0)", "2 = (y1=0, y2=1)",
                  "3 = (y1=1, y2=0)", "4 = (y1=1, y2=1)"))

## End(Not run)
```

---

binom2.or *Bivariate Binary Regression with an Odds Ratio (Family Function)*

Description

Fits a Palmgren (bivariate odds-ratio model, or bivariate logistic regression) model to two binary responses. Actually, a bivariate logistic/probit/cloglog/cauchit model can be fitted. The odds ratio is used as a measure of dependency.

Usage

```
binom2.or(lmu = "logitlink", lmu1 = lmu, lmu2 = lmu, loratio = "loglink",
          imu1 = NULL, imu2 = NULL, ioratio = NULL, zero = "oratio",
          exchangeable = FALSE, tol = 0.001, more.robust = FALSE)
```

Arguments

- `lmu` — Link function applied to the two marginal probabilities. See `Links` for more choices. See the note below.
- `lmu1, lmu2` — Link function applied to the first and second of the two marginal probabilities.
- `loratio` — Link function applied to the odds ratio. See `Links` for more choices.
- `imu1, imu2, ioratio` — Optional initial values for the marginal probabilities and odds ratio. See `CommonVGAMffArguments` for more details. In general good initial values are often required so use these arguments if convergence failure occurs.
- `zero` — Which linear/additive predictor is modelled as an intercept only? The default is for the odds ratio. A `NULL` means none. See `CommonVGAMffArguments` for more details.
- `exchangeable` — Logical. If `TRUE`, the two marginal probabilities are constrained to be equal.
- `tol` — Tolerance for testing independence. Should be some small positive numerical value.
- `more.robust` — Logical. If `TRUE` then some measures are taken to compute the derivatives and working weights more robustly, i.e., in an attempt to avoid numerical problems. Currently this feature is not debugged if set `TRUE`.

Details

Also known informally as the *Palmgren model*, the bivariate logistic model is a full-likelihood based model defined as two logistic regressions plus `log(oratio) = eta3` where `eta3` is the third linear/additive predictor relating the odds ratio to explanatory variables. Explicitly, the default model is

$$logit[P(Y_j = 1)] = \eta_j, \quad j = 1, 2$$

for the marginals, and

$$\log[P(Y_{00} = 1)P(Y_{11} = 1)/(P(Y_{01} = 1)P(Y_{10} = 1))] = \eta_3,$$

specifies the dependency between the two responses. Here, the responses equal 1 for a success and a 0 for a failure, and the odds ratio is often written $\psi = p_{00}p_{11}/(p_{10}p_{01})$. The model is fitted by maximum likelihood estimation since the full likelihood is specified. The two binary responses are independent if and only if the odds ratio is unity, or equivalently, the log odds ratio is 0. Fisher scoring is implemented.

The default models η_3 as a single parameter only, i.e., an intercept-only model, but this can be circumvented by setting `zero = NULL` in order to model the odds ratio as a function of all the explanatory variables. The function `binom2.or()` can handle other probability link functions such as `probitlink`, `clogloglink` and `cauchitlink` links as well, so is quite general. In fact, the two marginal probabilities can each have a different link function. A similar model is the *bivariate probit model* (`binom2.rho`), which is based on a standard bivariate normal distribution, but the bivariate probit model is less interpretable and flexible.

The `exchangeable` argument should be used when the error structure is exchangeable, e.g., with eyes or ears data.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

When fitted, the `fitted.values` slot of the object contains the four joint probabilities, labelled as (Y_1, Y_2) = (0,0), (0,1), (1,0), (1,1), respectively. These estimated probabilities should be extracted with the `fitted` generic function.

Note

At present we call `binom2.or` families a *bivariate odds-ratio model*. The response should be either a 4-column matrix of counts (whose columns correspond to (Y_1, Y_2) = (0,0), (0,1), (1,0), (1,1) respectively), or a two-column matrix where each column has two distinct values, or a factor with four levels. The function `rbinom2.or` may be used to generate such data. Successful convergence requires at least one case of each of the four possible outcomes.

By default, a constant odds ratio is fitted because `zero = 3`. Set `zero = NULL` if you want the odds ratio to be modelled as a function of the explanatory variables; however, numerical problems are more likely to occur.

The argument `lmu`, which is actually redundant, is used for convenience and for upward compatibility: specifying `lmu` only means the link function will be applied to `lmu1` and `lmu2`. Users who want a different link function for each of the two marginal probabilities should use the `lmu1` and `lmu2` arguments, and the argument lmu is then ignored. It doesn't make sense to specify exchangeable = TRUE and have different link functions for the two marginal probabilities.

Regarding Yee and Dirnbock (2009), the xij (see vglm.control) argument enables environmental variables with different values at the two time points to be entered into an exchangeable binom2.or model. See the author's webpage for sample code.

Author(s)

Thomas W. Yee

References

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*, 2nd ed. London: Chapman & Hall.

le Cessie, S. and van Houwelingen, J. C. (1994). Logistic regression for correlated binary data. *Applied Statistics*, **43**, 95–108.

Palmgren, J. (1989). *Regression Models for Bivariate Binary Responses*. Technical Report no. 101, Department of Biostatistics, University of Washington, Seattle.

Yee, T. W. and Dirnbock, T. (2009). Models for analysing species' presence/absence data at two time points. Journal of Theoretical Biology, **259**(4), 684–694.

See Also

rbinom2.or, binom2.rho, loglinb2, loglinb3, zipebcom, coalminers, binomialff, logitlink, probitlink, clogloglink, cauchitlink.

Examples

```
# Fit the model in Table 6.7 in McCullagh and Nelder (1989)
coalminers <- transform(coalminers, Age = (age - 42) / 5)
fit <- vglm(cbind(nBnW, nBW, BnW, BW) ~ Age,
            binom2.or(zero = NULL), data = coalminers)
fitted(fit)
summary(fit)
coef(fit, matrix = TRUE)
c(weights(fit, type = "prior")) * fitted(fit)  # Table 6.8

## Not run:  with(coalminers, matplot(Age, fitted(fit), type = "l", las = 1,
                         xlab = "(age - 42) / 5", lwd = 2))
with(coalminers, matpoints(Age, depvar(fit), col=1:4))
legend(x = -4, y = 0.5, lty = 1:4, col = 1:4, lwd = 2,
       legend = c("1 = (Breathlessness=0, Wheeze=0)",
                  "2 = (Breathlessness=0, Wheeze=1)",
                  "3 = (Breathlessness=1, Wheeze=0)",
                  "4 = (Breathlessness=1, Wheeze=1)"))
## End(Not run)


# Another model: pet ownership
## Not run:  data(xs.nz, package = "VGAMdata")
```

```
# More homogeneous:
petdata <- subset(xs.nz, ethnicity == "European" & age < 70 &
                         sex == "M")
petdata <- na.omit(petdata[, c("cat", "dog", "age")])
summary(petdata)
with(petdata, table(cat, dog))  # Can compute the odds ratio

fit <- vgam(cbind((1-cat) * (1-dog), (1-cat) * dog,
                     cat  * (1-dog),    cat  * dog) ~ s(age, df = 5),
            binom2.or(zero =    3), data = petdata, trace = TRUE)
colSums(depvar(fit))
coef(fit, matrix = TRUE)

## End(Not run)

## Not run:  # Plot the estimated probabilities
ooo <- order(with(petdata, age))
matplot(with(petdata, age)[ooo], fitted(fit)[ooo, ], type = "l",
        xlab = "Age", ylab = "Probability", main = "Pet ownership",
        ylim = c(0, max(fitted(fit))), las = 1, lwd = 1.5)
legend("topleft", col=1:4, lty = 1:4, leg = c("no cat or dog ",
       "dog only", "cat only", "cat and dog"), lwd = 1.5)
## End(Not run)
```

Binom2.rho *Bivariate Probit Model*

Description

Density and random generation for a bivariate probit model. The correlation parameter rho is the measure of dependency.

Usage

```
rbinom2.rho(n, mu1,
  mu2 = if (exchangeable) mu1 else stop("argument 'mu2' not specified"),
  rho = 0, exchangeable = FALSE, twoCols = TRUE,
  colnames = if (twoCols) c("y1","y2") else c("00", "01", "10", "11"),
  ErrorCheck = TRUE)
dbinom2.rho(mu1,
  mu2 = if (exchangeable) mu1 else stop("'mu2' not specified"),
  rho = 0, exchangeable = FALSE,
  colnames = c("00", "01", "10", "11"), ErrorCheck = TRUE)
```

Arguments

n	number of observations. Same as in runif. The arguments mu1, mu2, rho are recycled to this value.

`mu1, mu2` The marginal probabilities. Only `mu1` is needed if `exchangeable = TRUE`. Values should be between 0 and 1.

`rho` The correlation parameter. Must be numeric and lie between -1 and 1. The default value of zero means the responses are uncorrelated.

`exchangeable` Logical. If `TRUE`, the two marginal probabilities are constrained to be equal.

`twoCols` Logical. If `TRUE`, then a $n \times 2$ matrix of 1s and 0s is returned. If `FALSE`, then a $n \times 4$ matrix of 1s and 0s is returned.

`colnames` The `dimnames` argument of `matrix` is assigned `list(NULL, colnames)`.

`ErrorCheck` Logical. Do some error checking of the input parameters?

Details

The function `rbinom2.rho` generates data coming from a bivariate probit model. The data might be fitted with the **VGAM** family function `binom2.rho`.

The function `dbinom2.rho` does not really compute the density (because that does not make sense here) but rather returns the four joint probabilities.

Value

The function `rbinom2.rho` returns either a 2 or 4 column matrix of 1s and 0s, depending on the argument `twoCols`.

The function `dbinom2.rho` returns a 4 column matrix of joint probabilities; each row adds up to unity.

Author(s)

T. W. Yee

See Also

`binom2.rho`.

Examples

```
(myrho <- rhobitlink(2, inverse = TRUE))  # Example 1
nn <- 2000
ymat <- rbinom2.rho(nn, mu1 = 0.8, rho = myrho, exch = TRUE)
(mytab <- table(ymat[, 1], ymat[, 2], dnn = c("Y1", "Y2")))
fit <- vglm(ymat ~ 1, binom2.rho(exch = TRUE))
coef(fit, matrix = TRUE)

bdata <- data.frame(x2 = sort(runif(nn)))  # Example 2
bdata <- transform(bdata, mu1 = probitlink(-2+4*x2, inv = TRUE),
                          mu2 = probitlink(-1+3*x2, inv = TRUE))
dmat <- with(bdata, dbinom2.rho(mu1, mu2, myrho))
ymat <- with(bdata, rbinom2.rho(nn, mu1, mu2, myrho))
fit2 <- vglm(ymat ~ x2, binom2.rho, data = bdata)
coef(fit2, matrix = TRUE)
## Not run:  matplot(with(bdata, x2), dmat, lty = 1:4, col = 1:4,
```

```
        type = "l", main = "Joint probabilities",
        ylim = 0:1, lwd = 2, ylab = "Probability")
legend(x = 0.25, y = 0.9, lty = 1:4, col = 1:4, lwd = 2,
       legend = c("1 = (y1=0, y2=0)", "2 = (y1=0, y2=1)",
                  "3 = (y1=1, y2=0)", "4 = (y1=1, y2=1)"))
## End(Not run)
```

---

binom2.rho *Bivariate Probit Regression*

Description

Fits a bivariate probit model to two binary responses. The correlation parameter rho is the measure of dependency.

Usage

```
binom2.rho(lmu = "probitlink", lrho = "rhobitlink",
           imu1 = NULL, imu2 = NULL,
           irho = NULL, imethod = 1, zero = "rho",
           exchangeable = FALSE, grho = seq(-0.95, 0.95, by = 0.05),
           nsimEIM = NULL)
binom2.Rho(rho = 0, imu1 = NULL, imu2 = NULL,
           exchangeable = FALSE, nsimEIM = NULL)
```

Arguments

`lmu` Link function applied to the marginal probabilities. Should be left alone.

`lrho` Link function applied to the ρ association parameter. See `Links` for more choices.

`imu1, imu2` Optional initial values for the two marginal probabilities. May be a vector.

`irho` Optional initial value for ρ. If given, this should lie between -1 and 1. See below for more comments.

`zero` Specifies which linear/additive predictors are modelled as intercept-only. A `NULL` means none. Numerically, the ρ parameter is easiest modelled as an intercept only, hence the default. See `CommonVGAMffArguments` for more information.

`exchangeable` Logical. If `TRUE`, the two marginal probabilities are constrained to be equal.

`imethod, nsimEIM, grho`
See `CommonVGAMffArguments` for more information. A value of at least 100 for `nsimEIM` is recommended; the larger the value the better.

`rho` Numeric vector. Values are recycled to the needed length, and ought to be in range, which is $(-1, 1)$.

Details

The *bivariate probit model* was one of the earliest regression models to handle two binary responses jointly. It has a probit link for each of the two marginal probabilities, and models the association between the responses by the ρ parameter of a standard bivariate normal distribution (with zero means and unit variances). One can think of the joint probabilities being $\Phi(\eta_1, \eta_2; \rho)$ where Φ is the cumulative distribution function of a standard bivariate normal distribution.

Explicitly, the default model is

$$probit[P(Y_j = 1)] = \eta_j, \quad j = 1, 2$$

for the marginals, and

$$rhobit[rho] = \eta_3.$$

The joint probability $P(Y_1 = 1, Y_2 = 1) = \Phi(\eta_1, \eta_2; \rho)$, and from these the other three joint probabilities are easily computed. The model is fitted by maximum likelihood estimation since the full likelihood is specified. Fisher scoring is implemented.

The default models η_3 as a single parameter only, i.e., an intercept-only model for rho, but this can be circumvented by setting `zero = NULL` in order to model rho as a function of all the explanatory variables.

The bivariate probit model should not be confused with a *bivariate logit model* with a probit link (see `binom2.or`). The latter uses the odds ratio to quantify the association. Actually, the bivariate logit model is recommended over the bivariate probit model because the odds ratio is a more natural way of measuring the association between two binary responses.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

When fitted, the `fitted.values` slot of the object contains the four joint probabilities, labelled as (Y_1, Y_2) = (0,0), (0,1), (1,0), (1,1), respectively.

Note

See `binom2.or` about the form of input the response should have.

By default, a constant ρ is fitted because `zero = "rho"`. Set `zero = NULL` if you want the ρ parameter to be modelled as a function of the explanatory variables. The value ρ lies in the interval $(-1, 1)$, therefore a `rhobitlink` link is default.

Converge problems can occur. If so, assign `irho` a range of values and monitor convergence (e.g., set `trace = TRUE`). Else try `imethod`. Practical experience shows that local solutions can occur, and that `irho` needs to be quite close to the (global) solution. Also, `imu1` and `imu2` may be used.

This help file is mainly about `binom2.rho()`. `binom2.Rho()` fits a bivariate probit model with *known* ρ. The inputted `rho` is saved in the `misc` slot of the fitted object, with `rho` as the component name.

In some econometrics applications (e.g., Freedman 2010, Freedman and Sekhon 2010) one response is used as an explanatory variable, e.g., a *recursive* binomial probit model. Such will not work here. Historically, the bivariate probit model was the first VGAM I ever wrote, based on Ashford and Sowden (1970). I don't think they ever thought of it either! Hence the criticisms raised go beyond the use of what was originally intended.

Author(s)

Thomas W. Yee

References

Ashford, J. R. and Sowden, R. R. (1970). Multi-variate probit analysis. *Biometrics*, **26**, 535–546.

Freedman, D. A. (2010). *Statistical Models and Causal Inference: a Dialogue with the Social Sciences*, Cambridge: Cambridge University Press.

Freedman, D. A. and Sekhon, J. S. (2010). Endogeneity in probit response models. *Political Analysis*, **18**, 138–150.

See Also

`rbinom2.rho`, `rhobitlink`, `pbinorm`, `binom2.or`, `loglinb2`, `coalminers`, `binomialff`, `rhobitlink`, `fisherzlink`.

Examples

```
coalminers <- transform(coalminers, Age = (age - 42) / 5)
fit <- vglm(cbind(nBnW, nBW, BnW, BW) ~ Age,
            binom2.rho, data = coalminers, trace = TRUE)
summary(fit)
coef(fit, matrix = TRUE)
```

`binomialff` *Binomial Family Function*

Description

Family function for fitting generalized linear models to binomial responses

Usage

```
binomialff(link = "logitlink", multiple.responses = FALSE,
        parallel = FALSE, zero = NULL, bred = FALSE, earg.link = FALSE)
```

Arguments

`link` Link function; see `Links` and `CommonVGAMffArguments` for more information.

`multiple.responses`
Multivariate response? If `TRUE`, then the response is interpreted as M independent binary responses, where M is the number of columns of the response matrix. In this case, the response matrix should have Q columns consisting of counts (successes), and the `weights` argument should have Q columns consisting of the number of trials (successes plus failures).

If `FALSE` and the response is a (2-column) matrix, then the number of successes is given in the first column, and the second column is the number of failures.

parallel
: A logical or formula. Used only if multiple.responses is TRUE. This argument allows for the parallelism assumption whereby the regression coefficients for a variable is constrained to be equal over the M linear/additive predictors. If parallel = TRUE then the constraint is not applied to the intercepts.

zero
: An integer-valued vector specifying which linear/additive predictors are modelled as intercepts only. The values must be from the set {1,2,...,M}, where M is the number of columns of the matrix response. See CommonVGAMffArguments for more information.

earg.link
: Details at CommonVGAMffArguments.

bred
: Details at CommonVGAMffArguments. Setting bred = TRUE should work for multiple responses (multiple.responses = TRUE) and all **VGAM** link functions; it has been tested for logitlink only (and it gives similar results to **brglm** but not identical), and further testing is required. One result from fitting bias reduced binary regression is that finite regression coefficients occur when the data is separable (see example below). Currently hdeff.vglm does not work when bred = TRUE.

Details

This function is largely to mimic binomial, however there are some differences.

When used with cqo and cao, it may be preferable to use the clogloglink link.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, vgam, rrvglm, cqo, and cao.

Warning

See the above note regarding bred.

The maximum likelihood estimate will not exist if the data is *completely separable* or *quasi-completely separable*. See Chapter 10 of Altman et al. (2004) for more details, and **safeBinaryRegression** and hdeff.vglm. Yet to do: add a sepcheck = TRUE, say, argument to further detect this problem and give an appropriate warning.

Note

If multiple.responses is FALSE (default) then the response can be of one of two formats: a factor (first level taken as failure), or a 2-column matrix (first column = successes) of counts. The argument weights in the modelling function can also be specified as any vector of positive values. In general, 1 means success and 0 means failure (to check, see the y slot of the fitted object). Note that a general vector of proportions of success is no longer accepted.

The notation M is used to denote the number of linear/additive predictors.

If multiple.responses is TRUE, then the matrix response can only be of one format: a matrix of 1's and 0's (1 = success).

Fisher scoring is used. This can sometimes fail to converge by oscillating between successive iterations (Ridout, 1990). See the example below.

Author(s)

Thomas W. Yee

References

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*, 2nd ed. London: Chapman & Hall.

Altman, M. and Gill, J. and McDonald, M. P. (2004). *Numerical Issues in Statistical Computing for the Social Scientist*, Hoboken, NJ, USA: Wiley-Interscience.

Ridout, M. S. (1990). Non-convergence of Fisher's method of scoring—a simple example. *GLIM Newsletter*, 20(6).

See Also

hdeff.vglm, Links, alogitlink, asinlink, rrvglm, cqo, cao, betabinomial, posbinomial, zibinomial, double.expbinomial, seq2binomial, amlbinomial, simplex, binomial, simulate.vlm, **safeBinaryRegression**, residualsvglm.

Examples

```
shunua <- hunua[sort.list(with(hunua, altitude)), ]  # Sort by altitude
fit <- vglm(agaaus ~ poly(altitude, 2), binomialff(link = clogloglink),
            data = shunua)
## Not run:
plot(agaaus ~ jitter(altitude), shunua, ylab = "Pr(Agaaus = 1)",
     main = "Presence/absence of Agathis australis", col = 4, las = 1)
with(shunua, lines(altitude, fitted(fit), col = "orange", lwd = 2))
## End(Not run)

# Fit two species simultaneously
fit2 <- vgam(cbind(agaaus, kniexc) ~ s(altitude),
             binomialff(multiple.responses = TRUE), data = shunua)
## Not run:
with(shunua, matplot(altitude, fitted(fit2), type = "l",
     main = "Two species response curves", las = 1))
## End(Not run)

# Shows that Fisher scoring can sometime fail. See Ridout (1990).
ridout <- data.frame(v = c(1000, 100, 10), r = c(4, 3, 3), n = rep(5, 3))
(ridout <- transform(ridout, logv = log(v)))
# The iterations oscillates between two local solutions:
glm.fail <- glm(r / n ~ offset(logv) + 1, weight = n,
               binomial(link = 'cloglog'), ridout, trace = TRUE)
coef(glm.fail)
# vglm()'s half-stepping ensures the MLE of -5.4007 is obtained:
vglm.ok <- vglm(cbind(r, n-r) ~ offset(logv) + 1,
               binomialff(link = clogloglink), ridout, trace = TRUE)
coef(vglm.ok)

# Separable data
```

```
set.seed(123)
threshold <- 0
bdata <- data.frame(x2 = sort(rnorm(nn <- 100)))
bdata <- transform(bdata, y1 = ifelse(x2 < threshold, 0, 1))
fit <- vglm(y1 ~ x2, binomialff(bred = TRUE),
            data = bdata, criter = "coef", trace = TRUE)
coef(fit, matrix = TRUE)  # Finite!!
summary(fit)
## Not run:  plot(depvar(fit) ~ x2, data = bdata, col = "blue", las = 1)
lines(fitted(fit) ~ x2, data = bdata, col = "orange")
abline(v = threshold, col = "gray", lty = "dashed")
## End(Not run)
```

Binorm *Bivariate Normal Distribution Cumulative Distribution Function*

Description

Density, cumulative distribution function and random generation for the bivariate normal distribution distribution.

Usage

```
dbinorm(x1, x2, mean1 = 0, mean2 = 0, var1 = 1, var2 = 1, cov12 = 0,
        log = FALSE)
pbinorm(q1, q2, mean1 = 0, mean2 = 0, var1 = 1, var2 = 1, cov12 = 0)
rbinorm(n,      mean1 = 0, mean2 = 0, var1 = 1, var2 = 1, cov12 = 0)
 pnorm2(x1, x2, mean1 = 0, mean2 = 0, var1 = 1, var2 = 1, cov12 = 0)
```

Arguments

x1, x2, q1, q2	vector of quantiles.
mean1, mean2, var1, var2, cov12	vector of means, variances and the covariance.
n	number of observations. Same as rnorm.
log	Logical. If log = TRUE then the logarithm of the density is returned.

Details

The default arguments correspond to the standard bivariate normal distribution with correlation parameter $\rho = 0$. That is, two independent standard normal distributions. Let sd1 (say) be sqrt(var1) and written σ_1, etc. Then the general formula for the correlation coefficient is $\rho = cov/(\sigma_1 \sigma_2)$ where cov is argument cov12. Thus if arguments var1 and var2 are left alone then cov12 can be inputted with ρ.

One can think of this function as an extension of pnorm to two dimensions, however note that the argument names have been changed for **VGAM** 0.9-1 onwards.

Value

dbinorm gives the density, pbinorm gives the cumulative distribution function, rbinorm generates random deviates (n by 2 matrix).

Warning

Being based on an approximation, the results of pbinorm() may be negative! Also, pnorm2() should be withdrawn soon; use pbinorm() instead because it is identical.

Note

For rbinorm(), if the ith variance-covariance matrix is not positive-definite then the ith row is all NAs.

References

pbinorm() is based on Donnelly (1973), the code was translated from FORTRAN to ratfor using struct, and then from ratfor to C manually. The function was originally called bivnor, and TWY only wrote a wrapper function.

Donnelly, T. G. (1973). Algorithm 462: Bivariate Normal Distribution. *Communications of the ACM*, **16**, 638.

See Also

pnorm, binormal, uninormal.

Examples

```
yvec <- c(-5, -1.96, 0, 1.96, 5)
ymat <- expand.grid(yvec, yvec)
cbind(ymat, pbinorm(ymat[, 1], ymat[, 2]))

## Not run:  rhovec <- seq(-0.95, 0.95, by = 0.01)
plot(rhovec, pbinorm(0, 0, cov12 = rhovec),
     type = "l", col = "blue", las = 1)
abline(v = 0, h = 0.25, col = "gray", lty = "dashed")
## End(Not run)
```

---

binormal *Bivariate Normal Distribution Family Function*

Description

Maximum likelihood estimation of the five parameters of a bivariate normal distribution.

Usage

```
binormal(lmean1 = "identitylink", lmean2 = "identitylink",
         lsd1   = "loglink",     lsd2   = "loglink",
         lrho   = "rhobitlink",
         imean1 = NULL,       imean2 = NULL,
         isd1   = NULL,       isd2   = NULL,
         irho   = NULL,       imethod = 1,
         eq.mean = FALSE,     eq.sd = FALSE,
         zero = c("sd", "rho"), rho.arg = NA)
```

Arguments

`lmean1, lmean2, lsd1, lsd2, lrho`
: Link functions applied to the means, standard deviations and `rho` parameters. See `Links` for more choices. Being positive quantities, a log link is the default for the standard deviations.

`imean1, imean2, isd1, isd2, irho, imethod, zero`
: See `CommonVGAMffArguments` for more information.

`eq.mean, eq.sd`
: Logical or formula. Constrains the means or the standard deviations to be equal.

`rho.arg`
: If ρ is known then this argument may be assigned the (scalar) value lying in $(-1, 1)$. The default is to estimate that parameter so that $M = 5$. If known, then other arguments such as `lrho` and `irho` are ignored, and `"rho"` is removed from `zero`.

Details

For the bivariate normal distribution, this fits a linear model (LM) to the means, and by default, the other parameters are intercept-only. The response should be a two-column matrix. The correlation parameter is `rho`, which lies between -1 and 1 (thus the `rhobitlink` link is a reasonable choice). The fitted means are returned as the fitted values, which is in the form of a two-column matrix. Fisher scoring is implemented.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Warning

This function may be renamed to `normal2()` or something like that at a later date.

Note

If both equal means and equal standard deviations are desired then use something like `constraints = list("(Intercept)" = matrix(c(1,1,0,0,0, 0,0,1,1,0 ,0,0,0,0,1), 5, 3))` and maybe `zero = NULL` etc.

Author(s)

T. W. Yee

See Also

uninormal, trinormal, pbinorm, bistudentt.

Examples

```
set.seed(123); nn <- 1000
bdata <- data.frame(x2 = runif(nn), x3 = runif(nn))
bdata <- transform(bdata, y1 = rnorm(nn, 1 + 2 * x2),
                          y2 = rnorm(nn, 3 + 4 * x2))
fit1 <- vglm(cbind(y1, y2) ~ x2,
             binormal(eq.sd = TRUE), data = bdata, trace = TRUE)
coef(fit1, matrix = TRUE)
constraints(fit1)
summary(fit1)

# Estimated P(Y1 <= y1, Y2 <= y2) under the fitted model
var1  <- loglink(2 * predict(fit1)[, "loglink(sd1)"], inv = TRUE)
var2  <- loglink(2 * predict(fit1)[, "loglink(sd2)"], inv = TRUE)
cov12 <- rhobitlink(predict(fit1)[, "rhobitlink(rho)"], inv = TRUE)
head(with(bdata, pbinorm(y1, y2,
                         mean1 = predict(fit1)[, "mean1"],
                         mean2 = predict(fit1)[, "mean2"],
                         var1 = var1, var2 = var2, cov12 = cov12)))
```

---

binormalcop *Gaussian Copula (Bivariate) Family Function*

Description

Estimate the correlation parameter of the (bivariate) Gaussian copula distribution by maximum likelihood estimation.

Usage

```
binormalcop(lrho = "rhobitlink", irho = NULL, imethod = 1,
            parallel = FALSE, zero = NULL)
```

Arguments

`lrho, irho, imethod`
: Details at CommonVGAMffArguments. See Links for more link function choices.

`parallel, zero`
: Details at CommonVGAMffArguments. If parallel = TRUE then the constraint is applied to the intercept too.

Details

The cumulative distribution function is

$$P(Y_1 \le y_1, Y_2 \le y_2) = \Phi_2(\Phi^{-1}(y_1), \Phi^{-1}(y_2); \rho)$$

for $-1 < \rho < 1$, Φ_2 is the cumulative distribution function of a standard bivariate normal (see `pbinorm`), and Φ is the cumulative distribution function of a standard univariate normal (see `pnorm`).

The support of the function is the interior of the unit square; however, values of 0 and/or 1 are not allowed. The marginal distributions are the standard uniform distributions. When $\rho = 0$ the random variables are independent.

This **VGAM** family function can handle multiple responses, for example, a six-column matrix where the first 2 columns is the first out of three responses, the next 2 columns being the next response, etc.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Note

The response matrix must have a multiple of two-columns. Currently, the fitted value is a matrix with the same number of columns and values equal to 0.5. This is because each marginal distribution corresponds to a standard uniform distribution.

This **VGAM** family function is fragile; each response must be in the interior of the unit square. Setting `crit = "coef"` is sometimes a good idea because inaccuracies in `pbinorm` might mean unnecessary half-stepping will occur near the solution.

Author(s)

T. W. Yee

References

Schepsmeier, U. and Stober, J. (2014). Derivatives and Fisher information of bivariate copulas. *Statistical Papers* **55**, 525–542.

See Also

`rbinormcop`, `pnorm`, `kendall.tau`.

Examples

```
nn <- 1000
ymat <- rbinormcop(nn, rho = rhobitlink(-0.9, inverse = TRUE))
bdata <- data.frame(y1 = ymat[, 1], y2 = ymat[, 2],
                    y3 = ymat[, 1], y4 = ymat[, 2],
                    x2 = runif(nn))
summary(bdata)
```

```
## Not run:  plot(ymat, col = "blue")
fit1 <-  # 2 responses, e.g., (y1,y2) is the 1st
  vglm(cbind(y1, y2, y3, y4) ~ 1, fam = binormalcop,
       crit = "coef",  # Sometimes a good idea
       data = bdata, trace = TRUE)
coef(fit1, matrix = TRUE)
Coef(fit1)
head(fitted(fit1))
summary(fit1)

# Another example; rho is a linear function of x2
bdata <- transform(bdata, rho = -0.5 + x2)
ymat <- rbinormcop(n = nn, rho = with(bdata, rho))
bdata <- transform(bdata, y5 = ymat[, 1], y6 = ymat[, 2])
fit2 <- vgam(cbind(y5, y6) ~ s(x2), data = bdata,
             binormalcop(lrho = "identitylink"), trace = TRUE)
## Not run: plot(fit2, lcol = "blue", scol = "orange", se = TRUE)
```

Binormcop *Gaussian Copula (Bivariate) Distribution*

Description

Density, distribution function, and random generation for the (one parameter) bivariate Gaussian copula distribution.

Usage

```
dbinormcop(x1, x2, rho = 0, log = FALSE)
pbinormcop(q1, q2, rho = 0)
rbinormcop(n, rho = 0)
```

Arguments

x1, x2, q1, q2	vector of quantiles. The x1 and x2 should be in the interval $(0, 1)$. Ditto for q1 and q2.
n	number of observations. Same as rnorm.
rho	the correlation parameter. Should be in the interval $(-1, 1)$.
log	Logical. If TRUE then the logarithm is returned.

Details

See binormalcop, the **VGAM** family functions for estimating the parameter by maximum likelihood estimation, for the formula of the cumulative distribution function and other details.

Value

dbinormcop gives the density, pbinormcop gives the distribution function, and rbinormcop generates random deviates (a two-column matrix).

Note

Yettodo: allow `x1` and/or `x2` to have values 1, and to allow any values for `x1` and/or `x2` to be outside the unit square.

Author(s)

T. W. Yee

See Also

`binormalcop`, `binormal`.

Examples

```
## Not run:  edge <- 0.01  # A small positive value
N <- 101; x <- seq(edge, 1.0 - edge, len = N); Rho <- 0.7
ox <- expand.grid(x, x)
zedd <- dbinormcop(ox[, 1], ox[, 2], rho = Rho, log = TRUE)
contour(x, x, matrix(zedd, N, N), col = "blue", labcex = 1.5)
zedd <- pbinormcop(ox[, 1], ox[, 2], rho = Rho)
contour(x, x, matrix(zedd, N, N), col = "blue", labcex = 1.5)

## End(Not run)
```

`Biplackett` *Plackett's Bivariate Copula*

Description

Density, distribution function, and random generation for the (one parameter) bivariate Plackett copula.

Usage

```
dbiplackcop(x1, x2, oratio, log = FALSE)
pbiplackcop(q1, q2, oratio)
rbiplackcop(n, oratio)
```

Arguments

`x1, x2, q1, q2`	vector of quantiles.
`n`	number of observations. Same as in `runif`.
`oratio`	the positive odds ratio ψ.
`log`	Logical. If `TRUE` then the logarithm is returned.

Details

See biplackettcop, the **VGAM** family functions for estimating the parameter by maximum likelihood estimation, for the formula of the cumulative distribution function and other details.

Value

dbiplackcop gives the density, pbiplackcop gives the distribution function, and rbiplackcop generates random deviates (a two-column matrix).

Author(s)

T. W. Yee

References

Mardia, K. V. (1967). Some contributions to contingency-type distributions. *Biometrika*, **54**, 235–249.

See Also

biplackettcop, bifrankcop.

Examples

```
## Not run:  N <- 101; oratio <- exp(1)
x <- seq(0.0, 1.0, len = N)
ox <- expand.grid(x, x)
zedd <- dbiplackcop(ox[, 1], ox[, 2], oratio = oratio)
contour(x, x, matrix(zedd, N, N), col = "blue")
zedd <- pbiplackcop(ox[, 1], ox[, 2], oratio = oratio)
contour(x, x, matrix(zedd, N, N), col = "blue")

plot(rr <- rbiplackcop(n = 3000, oratio = oratio))
par(mfrow = c(1, 2))
hist(rr[, 1])  # Should be uniform
hist(rr[, 2])  # Should be uniform

## End(Not run)
```

---

biplackettcop *Plackett's Bivariate Copula Family Function*

Description

Estimate the association parameter of Plackett's bivariate distribution (copula) by maximum likelihood estimation.

Usage

```
biplackettcop(link = "loglink", ioratio = NULL, imethod = 1,
              nsimEIM = 200)
```

Arguments

- `link`: Link function applied to the (positive) odds ratio ψ. See `Links` for more choices and information.
- `ioratio`: Numeric. Optional initial value for ψ. If a convergence failure occurs try assigning a value or a different value.
- `imethod, nsimEIM`: See `CommonVGAMffArguments`.

Details

The defining equation is

$$\psi = H \times (1 - y_1 - y_2 + H)/((y_1 - H) \times (y_2 - H))$$

where $P(Y_1 \leq y_1, Y_2 \leq y_2) = H_\psi(y_1, y_2)$ is the cumulative distribution function. The density function is $h_\psi(y_1, y_2) =$

$$\psi[1 + (\psi - 1)(y_1 + y_2 - 2y_1 y_2)]/\left([1 + (\psi - 1)(y_1 + y_2)]^2 - 4\psi(\psi - 1)y_1 y_2\right)^{3/2}$$

for $\psi > 0$. Some writers call ψ the *cross product ratio* but it is called the *odds ratio* here. The support of the function is the unit square. The marginal distributions here are the standard uniform although it is commonly generalized to other distributions.

If $\psi = 1$ then $h_\psi(y_1, y_2) = y_1 y_2$, i.e., independence. As the odds ratio tends to infinity one has $y_1 = y_2$. As the odds ratio tends to 0 one has $y_2 = 1 - y_1$.

Fisher scoring is implemented using `rbiplackcop`. Convergence is often quite slow.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Note

The response must be a two-column matrix. Currently, the fitted value is a 2-column matrix with 0.5 values because the marginal distributions correspond to a standard uniform distribution.

Author(s)

T. W. Yee

References

Plackett, R. L. (1965). A class of bivariate distributions. *Journal of the American Statistical Association*, **60**, 516–522.

See Also

rbiplackcop, bifrankcop.

Examples

```
## Not run:
ymat <- rbiplackcop(n = 2000, oratio = exp(2))
plot(ymat, col = "blue")
fit <- vglm(ymat ~ 1, fam = biplackettcop, trace = TRUE)
coef(fit, matrix = TRUE)
Coef(fit)
vcov(fit)
head(fitted(fit))
summary(fit)

## End(Not run)
```

---

biplot-methods *Biplot of Constrained Regression Models*

Description

biplot is a generic function applied to RR-VGLMs and QRR-VGLMs etc. These apply to rank-1 and rank-2 models of these only. For RR-VGLMs these plot the second latent variable scores against the first latent variable scores.

Methods

x The object from which the latent variables are extracted and/or plotted.

Note

See lvplot which is very much related to biplots.

Bisa *The Birnbaum-Saunders Distribution*

Description

Density, distribution function, and random generation for the Birnbaum-Saunders distribution.

Usage

```
dbisa(x, scale = 1, shape, log = FALSE)
pbisa(q, scale = 1, shape, lower.tail = TRUE, log.p = FALSE)
qbisa(p, scale = 1, shape, lower.tail = TRUE, log.p = FALSE)
rbisa(n, scale = 1, shape)
```

Arguments

- `x, q` vector of quantiles.
- `p` vector of probabilities.
- `n` Same as in `runif`.
- `scale, shape` the (positive) scale and shape parameters.
- `log` Logical. If `TRUE` then the logarithm of the density is returned.
- `lower.tail, log.p` Same meaning as in `pnorm` or `qnorm`.

Details

The Birnbaum-Saunders distribution is a distribution which is used in survival analysis. See `bisa`, the **VGAM** family function for estimating the parameters, for more details.

Value

`dbisa` gives the density, `pbisa` gives the distribution function, and `qbisa` gives the quantile function, and `rbisa` generates random deviates.

Author(s)

T. W. Yee and Kai Huang

See Also

`bisa`.

Examples

```
## Not run:
x <- seq(0, 6, len = 400)
plot(x, dbisa(x, shape = 1), type = "l", col = "blue",
     ylab = "Density", lwd = 2, ylim = c(0,1.3), lty = 3,
     main = "X ~ Birnbaum-Saunders(shape, scale = 1)")
lines(x, dbisa(x, shape = 2), col = "orange", lty = 2, lwd = 2)
lines(x, dbisa(x, shape = 0.5), col = "green", lty = 1, lwd = 2)
legend(x = 3, y = 0.9, legend = paste("shape  = ",c(0.5, 1,2)),
       col = c("green","blue","orange"), lty = 1:3, lwd = 2)

shape <- 1; x <- seq(0.0, 4, len = 401)
plot(x, dbisa(x, shape = shape), type = "l", col = "blue",
     main = "Blue is density, orange is the CDF", las = 1,
     sub = "Red lines are the 10,20,...,90 percentiles",
     ylab = "", ylim = 0:1)
abline(h = 0, col = "blue", lty = 2)
lines(x, pbisa(x, shape = shape), col = "orange")
probs <- seq(0.1, 0.9, by = 0.1)
Q <- qbisa(probs, shape = shape)
lines(Q, dbisa(Q, shape = shape), col = "red", lty = 3, type = "h")
```

```
pbisa(Q, shape = shape) - probs  # Should be all zero
abline(h = probs, col = "red", lty = 3)
lines(Q, pbisa(Q, shape = shape), col = "red", lty = 3, type = "h")

## End(Not run)
```

---

bisa *Birnbaum-Saunders Regression Family Function*

Description

Estimates the shape and scale parameters of the Birnbaum-Saunders distribution by maximum likelihood estimation.

Usage

```
bisa(lscale = "loglink", lshape = "loglink", iscale = 1,
     ishape = NULL, imethod = 1, zero = "shape", nowarning = FALSE)
```

Arguments

`nowarning`	Logical. Suppress a warning? Ignored for **VGAM** 0.9-7 and higher.
`lscale, lshape`	Parameter link functions applied to the shape and scale parameters (a and b below). See `Links` for more choices. A log link is the default for both because they are positive.
`iscale, ishape`	Initial values for a and b. A `NULL` means an initial value is chosen internally using `imethod`.
`imethod`	An integer with value `1` or `2` or `3` which specifies the initialization method. If failure to converge occurs try the other value, or else specify a value for `ishape` and/or `iscale`.
`zero`	Specifies which linear/additive predictor is modelled as intercept-only. If used, choose one value from the set {1,2}. See `CommonVGAMffArguments` for more details.

Details

The (two-parameter) Birnbaum-Saunders distribution has a cumulative distribution function that can be written as

$$F(y; a, b) = \Phi[\xi(y/b)/a]$$

where $\Phi(\cdot)$ is the cumulative distribution function of a standard normal (see `pnorm`), $\xi(t) = \sqrt{t} - 1/\sqrt{t}$, $y > 0$, $a > 0$ is the shape parameter, $b > 0$ is the scale parameter. The mean of Y (which is the fitted value) is $b(1 + a^2/2)$. and the variance is $a^2 b^2 (1 + \frac{5}{4} a^2)$. By default, $\eta_1 = \log(a)$ and $\eta_2 = \log(b)$ for this family function.

Note that a and b are orthogonal, i.e., the Fisher information matrix is diagonal. This family function implements Fisher scoring, and it is unnecessary to compute any integrals numerically.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Author(s)

T. W. Yee

References

Lemonte, A. J. and Cribari-Neto, F. and Vasconcellos, K. L. P. (2007). Improved statistical inference for the two-parameter Birnbaum-Saunders distribution. *Computational Statistics & Data Analysis*, **51**, 4656–4681.

Birnbaum, Z. W. and Saunders, S. C. (1969). A new family of life distributions. *Journal of Applied Probability*, **6**, 319–327.

Birnbaum, Z. W. and Saunders, S. C. (1969). Estimation for a family of life distributions with applications to fatigue. *Journal of Applied Probability*, **6**, 328–347.

Engelhardt, M. and Bain, L. J. and Wright, F. T. (1981). Inferences on the parameters of the Birnbaum-Saunders fatigue life distribution based on maximum likelihood estimation. *Technometrics*, **23**, 251–256.

Johnson, N. L. and Kotz, S. and Balakrishnan, N. (1995). *Continuous Univariate Distributions*, 2nd edition, Volume 2, New York: Wiley.

See Also

pbisa, inv.gaussianff, CommonVGAMffArguments.

Examples

```
bdata1 <- data.frame(x2 = runif(nn <- 1000))
bdata1 <- transform(bdata1, shape = exp(-0.5 + x2),
                            scale = exp(1.5))
bdata1 <- transform(bdata1, y = rbisa(nn, scale, shape))
fit1 <- vglm(y ~ x2, bisa(zero = 1), data = bdata1, trace = TRUE)
coef(fit1, matrix = TRUE)

## Not run:
bdata2 <- data.frame(shape = exp(-0.5), scale = exp(0.5))
bdata2 <- transform(bdata2, y = rbisa(nn, scale, shape))
fit <- vglm(y ~ 1, bisa, data = bdata2, trace = TRUE)
with(bdata2, hist(y, prob = TRUE, ylim = c(0, 0.5),
                  col = "lightblue"))
coef(fit, matrix = TRUE)
with(bdata2, mean(y))
head(fitted(fit))
x <- with(bdata2, seq(0, max(y), len = 200))
lines(dbisa(x, Coef(fit)[1], Coef(fit)[2]) ~ x, data = bdata2,
      col = "orange", lwd = 2)
## End(Not run)
```

Bistudentt — *Bivariate Student-t Distribution Density Function*

Description

Density for the bivariate Student-t distribution.

Usage

```
dbistudentt(x1, x2, df, rho = 0, log = FALSE)
```

Arguments

x1, x2	vector of quantiles.
df, rho	vector of degrees of freedom and correlation parameter. For df, a value Inf is currently not working.
log	Logical. If log = TRUE then the logarithm of the density is returned.

Details

One can think of this function as an extension of dt to two dimensions. See bistudentt for more information.

Value

dbistudentt gives the density.

See Also

bistudentt, dt.

Examples

```
## Not run:  N <- 101; x <- seq(-4, 4, len = N); Rho <- 0.7
mydf <- 10; ox <- expand.grid(x, x)
zedd <- dbistudentt(ox[, 1], ox[, 2], df = mydf,
                    rho = Rho, log = TRUE)
contour(x, x, matrix(zedd, N, N), col = "blue", labcex = 1.5)

## End(Not run)
```

`bistudentt` *Bivariate Student-t Family Function*

Description

Estimate the degrees of freedom and correlation parameters of the (bivariate) Student-t distribution by maximum likelihood estimation.

Usage

```
bistudentt(ldf = "logloglink", lrho = "rhobitlink",
           idf = NULL, irho = NULL, imethod = 1,
           parallel = FALSE, zero = "rho")
```

Arguments

`ldf, lrho, idf, irho, imethod`
: Details at `CommonVGAMffArguments`. See `Links` for more link function choices.

`parallel, zero`
: Details at `CommonVGAMffArguments`.

Details

The density function is

$$f(y_1, y_2; \nu, \rho) = \frac{1}{2\pi\sqrt{1-\rho^2}}(1 + (y_1^2 + y_2^2 - 2\rho y_1 y_2)/(\nu(1-\rho^2)))^{-(\nu+2)/2}$$

for $-1 < \rho < 1$, and real y_1 and y_2.

This **VGAM** family function can handle multiple responses, for example, a six-column matrix where the first 2 columns is the first out of three responses, the next 2 columns being the next response, etc.

Value

An object of class "vglmff" (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Warning

The working weight matrices have not been fully checked.

Note

The response matrix must have a multiple of two-columns. Currently, the fitted value is a matrix with the same number of columns and values equal to 0.0.

Author(s)

T. W. Yee, with help from Thibault Vatter.

References

Schepsmeier, U. and Stober, J. (2014). Derivatives and Fisher information of bivariate copulas. *Statistical Papers* **55**, 525–542.

See Also

dbistudentt, binormal, pt.

Examples

```
nn <- 1000
mydof <- logloglink(1, inverse = TRUE)
ymat <- cbind(rt(nn, df = mydof), rt(nn, df = mydof))
bdata <- data.frame(y1 = ymat[, 1], y2 = ymat[, 2],
                    y3 = ymat[, 1], y4 = ymat[, 2],
                    x2 = runif(nn))
summary(bdata)
## Not run:  plot(ymat, col = "blue")
fit1 <-    # 2 responses, e.g., (y1,y2) is the 1st
  vglm(cbind(y1, y2, y3, y4) ~ 1,
       bistudentt,  # crit = "coef",  # Sometimes a good idea
       data = bdata, trace = TRUE)
coef(fit1, matrix = TRUE)
Coef(fit1)
head(fitted(fit1))
summary(fit1)
```

---

bmi.nz *Body Mass Index of New Zealand Adults Data*

Description

The body mass indexes and ages from an approximate random sample of 700 New Zealand adults.

Usage

```
data(bmi.nz)
```

Format

A data frame with 700 observations on the following 2 variables.

age a numeric vector; their age (years).

BMI a numeric vector; their body mass indexes, which is their weight divided by the square of their height (kg / m^2).

Details

They are a random sample from the Fletcher Challenge/Auckland Heart and Health survey conducted in the early 1990s.

There are some outliers in the data set.

A variable `gender` would be useful, and may be added later.

Source

Formerly the Clinical Trials Research Unit, University of Auckland, New Zealand.

References

MacMahon, S., Norton, R., Jackson, R., Mackie, M. J., Cheng, A., Vander Hoorn, S., Milne, A., McCulloch, A. (1995) Fletcher Challenge-University of Auckland Heart & Health Study: design and baseline findings. *New Zealand Medical Journal*, **108**, 499–502.

Examples

```
## Not run:  with(bmi.nz, plot(age, BMI, col = "blue"))
fit <- vgam(BMI ~ s(age, df = c(2, 4, 2)), lms.yjn,
            data = bmi.nz, trace = TRUE)
qtplot(fit, pcol = "blue", tcol = "brown", lcol = "brown")
## End(Not run)
```

`borel.tanner` *Borel-Tanner Distribution Family Function*

Description

Estimates the parameter of a Borel-Tanner distribution by maximum likelihood estimation.

Usage

```
borel.tanner(Qsize = 1, link = "logitlink", imethod = 1)
```

Arguments

`Qsize`	A positive integer. It is called Q below and is the initial queue size. The default value $Q = 1$ corresponds to the Borel distribution.
`link`	Link function for the parameter; see `Links` for more choices and for general information.
`imethod`	See `CommonVGAMffArguments`. Valid values are 1, 2, 3 or 4.

Details

The Borel-Tanner distribution (Tanner, 1953) describes the distribution of the total number of customers served before a queue vanishes given a single queue with random arrival times of customers (at a constant rate r per unit time, and each customer taking a constant time b to be served). Initially the queue has Q people and the first one starts to be served. The two parameters appear in the density only in the form of the product rb, therefore we use $a = rb$, say, to denote the single parameter to be estimated. The density function is

$$f(y; a) = \frac{Q}{(y - Q)!} y^{y-Q-1} a^{y-Q} \exp(-ay)$$

where $y = Q, Q + 1, Q + 2, \ldots$. The case $Q = 1$ corresponds to the *Borel* distribution (Borel, 1942). For the $Q = 1$ case it is necessary for $0 < a < 1$ for the distribution to be proper. The Borel distribution is a basic Lagrangian distribution of the first kind. The Borel-Tanner distribution is an Q-fold convolution of the Borel distribution.

The mean is $Q/(1 - a)$ (returned as the fitted values) and the variance is $Qa/(1 - a)^3$. The distribution has a very long tail unless a is small. Fisher scoring is implemented.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Author(s)

T. W. Yee

References

Tanner, J. C. (1953). A problem of interference between two queues. *Biometrika*, **40**, 58–69.

Borel, E. (1942). Sur l'emploi du theoreme de Bernoulli pour faciliter le calcul d'une infinite de coefficients. Application au probleme de l'attente a un guichet. *Comptes Rendus, Academie des Sciences, Paris, Series A*, **214**, 452–456.

Johnson N. L., Kemp, A. W. and Kotz S. (2005). *Univariate Discrete Distributions*, 3rd edition, p.328. Hoboken, New Jersey: Wiley.

Consul, P. C. and Famoye, F. (2006). *Lagrangian Probability Distributions*, Boston, MA, USA: Birkhauser.

See Also

`rbort`, `poissonff`, `felix`.

Examples

```
bdata <- data.frame(y = rbort(n <- 200))
fit <- vglm(y ~ 1, borel.tanner, bdata, trace = TRUE, crit = "c")
coef(fit, matrix = TRUE)
Coef(fit)
summary(fit)
```

Bort *The Borel-Tanner Distribution*

Description

Density and random generation for the Borel-Tanner distribution.

Usage

```
dbort(x, Qsize = 1, a = 0.5, log = FALSE)
rbort(n, Qsize = 1, a = 0.5)
```

Arguments

`x`	vector of quantiles.
`n`	number of observations. Must be a positive integer of length 1.
`Qsize, a`	See `borel.tanner`.
`log`	Logical. If `log = TRUE` then the logarithm of the density is returned.

Details

See `borel.tanner`, the **VGAM** family function for estimating the parameter, for the formula of the probability density function and other details.

Value

`dbort` gives the density, `rbort` generates random deviates.

Warning

Looping is used for `rbort`, therefore values of `a` close to 1 will result in long (or infinite!) computational times. The default value of `a` is subjective.

Author(s)

T. W. Yee

See Also

`borel.tanner`.

Examples

```
## Not run:  qsize <- 1; a <- 0.5; x <- qsize:(qsize+10)
plot(x, dbort(x, qsize, a), type = "h", las = 1, col = "blue",
     ylab = paste("fbort(qsize=", qsize, ", a=", a, ")"),
     log = "y", main = "Borel-Tanner density function")
## End(Not run)
```

Brat *Inputting Data to fit a Bradley Terry Model*

Description

Takes in a square matrix of counts and outputs them in a form that is accessible to the brat and bratt family functions.

Usage

```
Brat(mat, ties = 0 * mat, string = c(">", "=="), whitespace = FALSE)
```

Arguments

mat	Matrix of counts, which is considered M by M in dimension when there are ties, and $M+1$ by $M+1$ when there are no ties. The rows are winners and the columns are losers, e.g., the 2-1 element is now many times Competitor 2 has beaten Competitor 1. The matrices are best labelled with the competitors' names.
ties	Matrix of counts. This should be the same dimension as `mat`. By default, there are no ties. The matrix must be symmetric, and the diagonal should contain `NA`s.
string	Character. The matrices are labelled with the first value of the descriptor, e.g., `"NZ > Oz"` 'means' NZ beats Australia in rugby. Suggested alternatives include `" beats "` or `" wins against "`. The second value is used to handle ties.
whitespace	Logical. If `TRUE` then a white space is added before and after `string`; it generally enhances readability. See CommonVGAMffArguments for some similar-type information.

Details

In the **VGAM** package it is necessary for each matrix to be represented as a single row of data by brat and bratt. Hence the non-diagonal elements of the $M+1$ by $M+1$ matrix are concatenated into $M(M+1)$ values (no ties), while if there are ties, the non-diagonal elements of the M by M matrix are concatenated into $M(M-1)$ values.

Value

A matrix with 1 row and either $M(M+1)$ or $M(M-1)$ columns.

Note

This is a data preprocessing function for brat and bratt.

Yet to do: merge `InverseBrat` into `brat`.

Author(s)

T. W. Yee

References

Agresti, A. (2013). *Categorical Data Analysis*, 3rd ed. Hoboken, NJ, USA: Wiley.

See Also

brat, bratt, InverseBrat.

Examples

```
journal <- c("Biometrika", "Comm Statist", "JASA", "JRSS-B")
mat <- matrix(c( NA, 33, 320, 284,   730, NA, 813, 276,
                498, 68,  NA, 325,   221, 17, 142, NA), 4, 4)
dimnames(mat) <- list(winner = journal, loser = journal)
Brat(mat)  # Less readable
Brat(mat, whitespace = TRUE)  # More readable
vglm(Brat(mat, whitespace = TRUE) ~ 1, brat, trace = TRUE)
```

---

brat *Bradley Terry Model*

Description

Fits a Bradley Terry model (intercept-only model) by maximum likelihood estimation.

Usage

```
brat(refgp = "last", refvalue = 1, ialpha = 1)
```

Arguments

refgp	Integer whose value must be from the set $\{1,\ldots,M+1\}$, where there are $M+1$ competitors. The default value indicates the last competitor is used—but don't input a character string, in general.
refvalue	Numeric. A positive value for the reference group.
ialpha	Initial values for the αs. These are recycled to the appropriate length.

Details

The Bradley Terry model involves $M+1$ competitors who either win or lose against each other (no draws/ties allowed in this implementation–see bratt if there are ties). The probability that Competitor i beats Competitor j is $\alpha_i/(\alpha_i+\alpha_j)$, where all the αs are positive. Loosely, the αs can be thought of as the competitors' 'abilities'. For identifiability, one of the α_i is set to a known value refvalue, e.g., 1. By default, this function chooses the last competitor to have this reference value. The data can be represented in the form of a $M+1$ by $M+1$ matrix of counts, where winners are the rows and losers are the columns. However, this is not the way the data should be inputted (see below).

Excluding the reference value/group, this function chooses $\log(\alpha_j)$ as the M linear predictors. The log link ensures that the αs are positive.

The Bradley Terry model can be fitted by logistic regression, but this approach is not taken here. The Bradley Terry model can be fitted with covariates, e.g., a home advantage variable, but unfortunately, this lies outside the VGLM theoretical framework and therefore cannot be handled with this code.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`.

Warning

Presently, the residuals are wrong, and the prior weights are not handled correctly. Ideally, the total number of counts should be the prior weights, after the response has been converted to proportions. This would make it similar to family functions such as `multinomial` and `binomialff`.

Note

The function `Brat` is useful for coercing a $M + 1$ by $M + 1$ matrix of counts into a one-row matrix suitable for `brat`. Diagonal elements are skipped, and the usual S order of `c(a.matrix)` of elements is used. There should be no missing values apart from the diagonal elements of the square matrix. The matrix should have winners as the rows, and losers as the columns. In general, the response should be a 1-row matrix with $M(M + 1)$ columns.

Only an intercept model is recommended with `brat`. It doesn't make sense really to include covariates because of the limited VGLM framework.

Notationally, note that the **VGAM** family function `brat` has $M + 1$ contestants, while `bratt` has M contestants.

Author(s)

T. W. Yee

References

Agresti, A. (2013). *Categorical Data Analysis*, 3rd ed. Hoboken, NJ, USA: Wiley.

Stigler, S. (1994). Citation patterns in the journals of statistics and probability. *Statistical Science*, **9**, 94–108.

The **BradleyTerry2** package has more comprehensive capabilities than this function.

See Also

`bratt`, `Brat`, `multinomial`, `binomialff`.

Examples

```
# Citation statistics: being cited is a 'win'; citing is a 'loss'
journal <- c("Biometrika", "Comm.Statist", "JASA", "JRSS-B")
mat <- matrix(c( NA, 33, 320, 284,
                730, NA, 813, 276,
                498, 68,  NA, 325,
                221, 17, 142,  NA), 4, 4)
dimnames(mat) <- list(winner = journal, loser = journal)
fit <- vglm(Brat(mat) ~ 1, brat(refgp = 1), trace = TRUE)
fit <- vglm(Brat(mat) ~ 1, brat(refgp = 1), trace = TRUE, crit = "coef")
summary(fit)
c(0, coef(fit))  # Log-abilities (in order of "journal")
c(1, Coef(fit))  # Abilities (in order of "journal")
fitted(fit)     # Probabilities of winning in awkward form
(check <- InverseBrat(fitted(fit)))  # Probabilities of winning
check + t(check)  # Should be 1's in the off-diagonals
```

---

bratt *Bradley Terry Model With Ties*

Description

Fits a Bradley Terry model with ties (intercept-only model) by maximum likelihood estimation.

Usage

```
bratt(refgp = "last", refvalue = 1, ialpha = 1, i0 = 0.01)
```

Arguments

refgp	Integer whose value must be from the set $\{1,\ldots,M\}$, where there are M competitors. The default value indicates the last competitor is used—but don't input a character string, in general.
refvalue	Numeric. A positive value for the reference group.
ialpha	Initial values for the αs. These are recycled to the appropriate length.
i0	Initial value for α_0. If convergence fails, try another positive value.

Details

There are several models that extend the ordinary Bradley Terry model to handle ties. This family function implements one of these models. It involves M competitors who either win or lose or tie against each other. (If there are no draws/ties then use brat). The probability that Competitor i beats Competitor j is $\alpha_i/(\alpha_i + \alpha_j + \alpha_0)$, where all the αs are positive. The probability that Competitor i ties with Competitor j is $\alpha_0/(\alpha_i + \alpha_j + \alpha_0)$. Loosely, the αs can be thought of as the competitors' 'abilities', and α_0 is an added parameter to model ties. For identifiability, one of the α_i is set to a known value refvalue, e.g., 1. By default, this function chooses the last competitor to have this reference value. The data can be represented in the form of a M by M matrix of counts, where winners are the rows and losers are the columns. However, this is not the way the data should be inputted (see below).

Excluding the reference value/group, this function chooses $\log(\alpha_j)$ as the first $M-1$ linear predictors. The log link ensures that the αs are positive. The last linear predictor is $\log(\alpha_0)$.

The Bradley Terry model can be fitted with covariates, e.g., a home advantage variable, but unfortunately, this lies outside the VGLM theoretical framework and therefore cannot be handled with this code.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm.

Note

The function Brat is useful for coercing a M by M matrix of counts into a one-row matrix suitable for bratt. Diagonal elements are skipped, and the usual S order of c(a.matrix) of elements is used. There should be no missing values apart from the diagonal elements of the square matrix. The matrix should have winners as the rows, and losers as the columns. In general, the response should be a matrix with $M(M-1)$ columns.

Also, a symmetric matrix of ties should be passed into Brat. The diagonal of this matrix should be all NAs.

Only an intercept model is recommended with bratt. It doesn't make sense really to include covariates because of the limited VGLM framework.

Notationally, note that the **VGAM** family function brat has $M+1$ contestants, while bratt has M contestants.

Author(s)

T. W. Yee

References

Torsney, B. (2004). Fitting Bradley Terry models using a multiplicative algorithm. In: Antoch, J. (ed.) *Proceedings in Computational Statistics COMPSTAT 2004*, Physica-Verlag: Heidelberg. Pages 513–526.

See Also

brat, Brat, binomialff.

Examples

```
# citation statistics: being cited is a 'win'; citing is a 'loss'
journal <- c("Biometrika", "Comm.Statist", "JASA", "JRSS-B")
mat <- matrix(c( NA, 33, 320, 284,
                730, NA, 813, 276,
                498, 68,  NA, 325,
                221, 17, 142,  NA), 4, 4)
```

```
dimnames(mat) <- list(winner = journal, loser = journal)

# Add some ties. This is fictitional data.
ties <- 5 + 0 * mat
ties[2, 1] <- ties[1,2] <- 9

# Now fit the model
fit <- vglm(Brat(mat, ties) ~ 1, bratt(refgp = 1), trace = TRUE,
            crit = "coef")

summary(fit)
c(0, coef(fit))  # Log-abilities (last is log(alpha0))
c(1, Coef(fit))  #     Abilities (last is alpha0)

fit@misc$alpha   # alpha_1,...,alpha_M
fit@misc$alpha0  # alpha_0

fitted(fit)  # Probabilities of winning and tying, in awkward form
predict(fit)
(check <- InverseBrat(fitted(fit)))    # Probabilities of winning
qprob <- attr(fitted(fit), "probtie")  # Probabilities of a tie
qprobmat <- InverseBrat(c(qprob), NCo = nrow(ties))  # Pr(tie)
check + t(check) + qprobmat  # Should be 1s in the off-diagonals
```

---

budworm *Western Spuce Budworm*

Description

Counts of western spuce budworm (Choristoneura freemani) across seven developmental stages (five larval instars, pupae, and adults) on 12 sampling occasions.

Usage

```
data(budworm)
```

Format

A data frame with the following variables.

ddeg Degree days.

total Sum of stages 1–7.

stage1, stage2, stage3, stage4 Successive stages.

stage5, stage6, stage7 Successive stages.

Details

This data concerns the development of a defoliating moth widespread in western North America (i.e., north of Mexico). According to Boersch-Supan (2021), the insect passes hrough successive stages $j = 1, \dots, r$, delimited by $r - 1$ moults. The data was originally used in a 1986 publication but has been corrected for two sampling occasions; the data appears in Candy (1990) and was analyzed in Boersch-Supan (2021). See the latter for more references.

Source

Candy, S. G. (1990). *Biology of the mountain pinhole borer, Platypus subgranosus Scheld, in Tasmania*. MA thesis, University of Tasmania, Australia. `https://eprints.utas.edu.au/18864/`.

References

Boersch-Supan, P. H. (2021). Modeling insect phenology using ordinal regression and continuation ratio models. *ReScience C*, **7.1**, 1–14.

Examples

```
budworm
summary(budworm)
```

`calibrate` *Model Calibrations*

Description

`calibrate` is a generic function used to produce calibrations from various model fitting functions. The function invokes particular 'methods' which depend on the 'class' of the first argument.

Usage

```
calibrate(object, ...)
```

Arguments

`object`	An object for which a calibration is desired.
`...`	Additional arguments affecting the calibration produced. Usually the most important argument in `...` is `newdata` which, for `calibrate`, contains new *response* data, **Y**, say.

Details

Given a regression model with explanatory variables **X** and response **Y**, calibration involves estimating **X** from **Y** using the regression model. It can be loosely thought of as the opposite of `predict` (which takes an **X** and returns a **Y** of some sort.) In general, the central algorithm is maximum likelihood calibration.

Value

In general, given a new response **Y**, some function of the explanatory variables **X** are returned. For example, for constrained ordination models such as CQO and CAO models, it is usually not possible to return **X**, so the latent variables are returned instead (they are linear combinations of the **X**). See the specific `calibrate` methods functions to see what they return.

Note

This function was not called `predictx` because of the inability of constrained ordination models to return **X**; they can only return the latent variable values (also known as site scores) instead.

Author(s)

T. W. Yee

References

ter Braak, C. J. F. and van Dam, H. (1989). Inferring pH from diatoms: a comparison of old and new calibration methods. *Hydrobiologia*, **178**, 209–223.

See Also

`predict`, `calibrate.rrvglm`, `calibrate.qrrvglm`.

Examples

```
## Not run:
hspider[, 1:6] <- scale(hspider[, 1:6])  # Stdzed environmental vars
set.seed(123)
pcao1 <- cao(cbind(Pardlugu, Pardmont, Pardnigr, Pardpull, Zoraspin) ~
         WaterCon + BareSand + FallTwig + CoveMoss + CoveHerb + ReflLux,
         family = poissonff, data = hspider, Rank = 1, Bestof = 3,
         df1.nl = c(Zoraspin = 2, 1.9), Crow1positive = TRUE)

siteNos <- 1:2  # Calibrate these sites
cpcao1 <- calibrate(pcao1, trace = TRUE,
                    newdata = data.frame(depvar(pcao1)[siteNos, ],
                                         model.matrix(pcao1)[siteNos, ]))

# Graphically compare the actual site scores with their calibrated values
persp(pcao1, main = "Site scores: solid=actual, dashed=calibrated",
      label = TRUE, col = "blue", las = 1)
abline(v = latvar(pcao1)[siteNos], col = seq(siteNos))  # Actual scores
abline(v = cpcao1, lty = 2, col = seq(siteNos))  # Calibrated values

## End(Not run)
```

calibrate-methods *Calibration for Constrained Regression Models*

Description

calibrate is a generic function applied to RR-VGLMs, QRR-VGLMs and RR-VGAMs, etc.

Methods

object The object from which the calibration is performed.

calibrate.qrrvglm *Calibration for CQO and CAO models*

Description

Performs maximum likelihood calibration for constrained quadratic and additive ordination models (CQO and CAO models are better known as QRR-VGLMs and RR-VGAMs respectively).

Usage

```
calibrate.qrrvglm(object, newdata = NULL,
    type = c("latvar", "predictors", "response", "vcov", "everything"),
    lr.confint = FALSE, cf.confint = FALSE,
    level = 0.95, initial.vals = NULL, ...)
```

Arguments

object The fitted CQO/CAO model.

newdata A data frame with new response data, such as new species data. The default is to use the original data used to fit the model; however, the calibration may take a long time to compute because the computations are expensive.

Note that the creation of the model frame associated with newdata is fragile. Factors may not be created properly. If a variable is binary then its best for it to be straightforward and have only 0 and 1 as values.

type What type of result to be returned. The first are the calibrated latent variables or site scores. This is always computed. The "predictors" are the linear/quadratic or additive predictors evaluated at the calibrated latent variables or site scores. The "response" are the fitted values (usually means) evaluated at the calibrated latent variables or site scores. The "vcov" are the Wald-type estimated variance-covariance matrices of the calibrated latent variables or site scores. The "everything" is for all of them, i.e., all types. Note that for CAO models, the "vcov" type is unavailable.

`lr.confint, level`
: Compute *approximate* likelihood ratio based confidence intervals? If `TRUE` then `level` is the confidence level required and one should have `type = "latvar"` or `type = "everything"`; and currently only rank-1 models are supported. This option works for CLO and CQO models and not for CAO models. The function `uniroot` is called to solve for the root of a nonlinear equation to obtain each confidence limit, and this is not entirely reliable. It is assumed that the likelihood function is unimodal about its MLE because only one root is returned if there is more than one. One root is found on each side of the MLE. Technically, the default is to find the value of the latent variable whose difference in deviance (or twice the difference in log-likelihoods) from the optimal model is equal to `qchisq(level, df = 1)`. The intervals are not true profile likelihood intervals because it is not possible to estimate the regression coefficients of the QRR-VGLM/RR-VGLM based on one response vector. See `confint` to get the flavour of these two arguments in general.

`cf.confint`
: Compute *approximate* characteristic function based confidence intervals? If `TRUE` then `level` is the confidence level required and one should have `type = "latvar"` or `type = "everything"`; and currently only rank-1 models are supported. This option works for `binomialff` and `poissonff` CLO and CQO models and not for CAO models. The function `uniroot` is called to solve for the root of a nonlinear equation to obtain each confidence limit, and this is not entirely reliable. It is assumed that the likelihood function is unimodal because only one root is returned if there is more than one. Technically, the CDF of a normalized score statistic is obtained by Gauss–Hermite numerical integration of a complex-valued integrand, and this is based on the inversion formula described in Abate and Witt (1992).

`initial.vals`
: Initial values for the search. For rank-1 models, this should be a vector having length equal to `nrow(newdata)`, and for rank-2 models this should be a two-column matrix with the number of rows equalling the number of rows in `newdata`. The default is a grid defined by arguments in `calibrate.qrrvglm.control`.

`...`
: Arguments that are fed into `calibrate.qrrvglm.control`.

Details

Given a fitted regression CQO/CAO model, maximum likelihood calibration is theoretically easy and elegant. However, the method assumes that all the responses are independent, which is often not true in practice. More details and references are given in Yee (2018) and ch.6 of Yee (2015).

The function `optim` is used to search for the maximum likelihood solution. Good initial values are needed, and arguments in `calibrate.qrrvglm.control` allows the user some control over the choice of these.

Value

Several methods are implemented to obtain confidence intervals/regions for the calibration estimates. One method is when `lr.confint = TRUE`, then a 4-column matrix is returned with the confidence limits being the final 2 columns (if `type = "everything"` then the matrix is returned in the `lr.confint` list component). Another similar method is when `cf.confint = TRUE`. There may be some redundancy in whatever is returned. Other methods are returned by using `type` and they are described as follows.

The argument `type` determines what is returned. If `type = "everything"` then all the `type` values are returned in a list, with the following components. Each component has length `nrow(newdata)`.

`latvar` Calibrated latent variables or site scores (the default). This may have the attribute `"objectiveFunction"` which is usually the log-likelihood or the deviance.

`predictors` linear/quadratic or additive predictors. For example, for Poisson families, this will be on a log scale, and for binomial families, this will be on a logit scale.

`response` Fitted values of the response, evaluated at the calibrated latent variables.

`vcov` Wald-type estimated variance-covariance matrices of the calibrated latent variables or site scores. Actually, these are stored in a 3-D array whose dimension is `c(Rank(object), Rank(object), nrow(newdata))`. This type has only been implemented for `binomialff` and `poissonff` models with canonical links and `noRRR = ~ 1` and, for CQOs, `I.tolerances = TRUE` or `eq.tolerances = TRUE`.

Warning

This function is computationally expensive. Setting `trace = TRUE` to get a running log can be a good idea. This function has been tested but not extensively.

Note

Despite the name of this function, CAO models are handled as well to a certain extent. Some combinations of parameters are not handled, e.g., `lr.confint = TRUE` only works for rank-1, `type = "vcov"` only works for `binomialff` and `poissonff` models with canonical links and `noRRR = ~ 1`, and higher-order rank models need `eq.tolerances = TRUE` or `I.tolerances = TRUE` as well. For rank-1 objects, `lr.confint = TRUE` is recommended above `type = "vcov"` in terms of accuracy and overall generality. For class `"qrrvglm"` objects it is necessary that all response' tolerance matrices are positive-definite which correspond to bell-shaped response curves/surfaces.

For `binomialff` and `poissonff` models the `deviance` slot is used for the optimization rather than the `loglikelihood` slot, therefore one can calibrate using real-valued responses. (If the `loglikelihood` slot were used then functions such as `dpois` would be used with `log = TRUE` and then would be restricted to feed in integer-valued response values.)

Maximum likelihood calibration for Gaussian logit regression models may be performed by **rioja** but this applies to a single environmental variable such as `pH` in `data("SWAP", package = "rioja")`. In **VGAM** `calibrate()` estimates values of the *latent variable* rather than individual explanatory variables, hence the setting is more on ordination.

Author(s)

T. W. Yee. Recent work on the standard errors by David Zucker and Sam Oman at HUJI is gratefully acknowledged—these are returned in the `vcov` component and provided inspiration for `lr.confint` and `cf.confint`. A joint publication is being prepared on this subject.

References

Abate, J. and Whitt, W. (1992). The Fourier-series method for inverting transforms of probability distributions. *Queueing Systems*, **10**, 5–88.

ter Braak, C. J. F. (1995). Calibration. In: *Data Analysis in Community and Landscape Ecology* by Jongman, R. H. G., ter Braak, C. J. F. and van Tongeren, O. F. R. (Eds.) Cambridge University Press, Cambridge.

See Also

calibrate.qrrvglm.control, calibrate.rrvglm, calibrate, cqo, cao, optim, uniroot.

Examples

```
## Not run:
hspider[, 1:6] <- scale(hspider[, 1:6])  # Stdze environmental variables
set.seed(123)
siteNos <- c(1, 5)  # Calibrate these sites
pet1 <- cqo(cbind(Pardlugu, Pardmont, Pardnigr, Pardpull, Zoraspin) ~
        WaterCon + BareSand + FallTwig + CoveMoss + CoveHerb + ReflLux,
        trace = FALSE,
        data = hspider[-siteNos, ],  # Sites not in fitted model
        family = poissonff, I.toler = TRUE, Crow1positive = TRUE)
y0 <- hspider[siteNos, colnames(depvar(pet1))]  # Species counts
(cpet1 <- calibrate(pet1, trace = TRUE, newdata = data.frame(y0)))
(clrpet1 <- calibrate(pet1, lr.confint = TRUE, newdata = data.frame(y0)))
(ccfpet1 <- calibrate(pet1, cf.confint = TRUE, newdata = data.frame(y0)))
(cp1wald <- calibrate(pet1, newdata = y0, type = "everything"))

## End(Not run)

## Not run:
# Graphically compare the actual site scores with their calibrated
# values. 95 percent likelihood-based confidence intervals in green.
persp(pet1, main = "Site scores: solid=actual, dashed=calibrated",
      label = TRUE, col = "gray50", las = 1)
# Actual site scores:
xvars <- rownames(concoef(pet1))  # Variables comprising the latvar
est.latvar <- as.matrix(hspider[siteNos, xvars]) %*% concoef(pet1)
abline(v = est.latvar, col = seq(siteNos))
abline(v = cpet1, lty = 2, col = seq(siteNos))  # Calibrated values
arrows(clrpet1[,  3], c(60, 60), clrpet1[,  4], c(60, 60),  # Add CIs
       length = 0.08, col = "orange", angle = 90, code = 3, lwd = 2)
arrows(ccfpet1[,  3], c(70, 70), ccfpet1[,  4], c(70, 70),  # Add CIs
       length = 0.08, col = "limegreen", angle = 90, code = 3, lwd = 2)
arrows(cp1wald$latvar - 1.96 * sqrt(cp1wald$vcov), c(65, 65),
       cp1wald$latvar + 1.96 * sqrt(cp1wald$vcov), c(65, 65),  # Wald CIs
       length = 0.08, col = "blue", angle = 90, code = 3, lwd = 2)
legend("topright", lwd = 2,
       leg = c("CF interval", "Wald  interval", "LR interval"),
       col = c("limegreen", "blue", "orange"), lty = 1)
```

```
## End(Not run)
```

---

calibrate.qrrvglm.control

Control Function for CQO/CAO Calibration

Description

Algorithmic constants and parameters for running calibrate.qrrvglm are set using this function.

Usage

```
calibrate.qrrvglm.control(object, trace = FALSE, method.optim = "BFGS",
    gridSize = ifelse(Rank == 1, 21, 9), varI.latvar = FALSE, ...)
```

Arguments

object	The fitted CQO/CAO model. The user should ignore this argument.
trace	Logical indicating if output should be produced for each iteration. It is a good idea to set this argument to be TRUE since the computations are expensive.
method.optim	Character. Fed into the method argument of optim.
gridSize	Numeric, recycled to length Rank. Controls the resolution of the grid used for initial values. For each latent variable, an equally spaced grid of length gridSize is cast from the smallest site score to the largest site score. Then the likelihood function is evaluated on the grid, and the best fit is chosen as the initial value. Thus increasing the value of gridSize increases the chance of obtaining the global solution, however, the computing time increases proportionately.
varI.latvar	Logical. For CQO objects only, this argument is fed into Coef.qrrvglm.
...	Avoids an error message for extraneous arguments.

Details

Most CQO/CAO users will only need to make use of trace and gridSize. These arguments should be used inside their call to calibrate.qrrvglm, not this function directly.

Value

A list which with the following components.

trace	Numeric (even though the input can be logical).
gridSize	Positive integer.
varI.latvar	Logical.

Note

Despite the name of this function, CAO models are handled as well.

References

Yee, T. W. (2020). On constrained and unconstrained quadratic ordination. *Manuscript in preparation*.

See Also

calibrate.qrrvglm, Coef.qrrvglm.

Examples

```
## Not run:  hspider[, 1:6] <- scale(hspider[, 1:6])  # Needed for I.tol=TRUE
set.seed(123)
p1 <- cqo(cbind(Alopacce, Alopcune, Pardlugu, Pardnigr,
                Pardpull, Trocterr, Zoraspin) ~
          WaterCon + BareSand + FallTwig + CoveMoss + CoveHerb + ReflLux,
          family = poissonff, data = hspider, I.tol = TRUE)
sort(deviance(p1, history = TRUE))  # A history of all the iterations
siteNos <- 3:4  # Calibrate these sites
cp1 <- calibrate(p1, trace = TRUE,
                 new = data.frame(depvar(p1)[siteNos, ]))

## End(Not run)
## Not run:
# Graphically compare the actual site scores with their calibrated values
persp(p1, main = "Site scores: solid=actual, dashed=calibrated",
      label = TRUE, col = "blue", las = 1)
abline(v = latvar(p1)[siteNos], col = seq(siteNos))  # Actual site scores
abline(v = cp1, lty = 2, col = seq(siteNos))  # Calibrated values

## End(Not run)
```

---

calibrate.rrvglm *Calibration for CLO models (RR-VGLMs)*

Description

Performs maximum likelihood calibration for constrained linear ordination models (CLO models are better known as RR-VGLMs).

Usage

```
calibrate.rrvglm(object, newdata = NULL,
    type = c("latvar", "predictors", "response", "vcov", "everything"),
    lr.confint = FALSE, cf.confint = FALSE,
    level = 0.95, initial.vals = NULL, ...)
```

Arguments

object — The fitted `rrvglm` model. Note that `object` should be fitted with corner constraints.

newdata — See `calibrate.qrrvglm`.

type — See `calibrate.qrrvglm`. If `type = "vcov"` then `object` should have been fitted using `binomialff` or `poissonff` with canonical links, and have `noRRR = ~ 1`.

lr.confint, cf.confint, level — Same as `calibrate.qrrvglm`.

initial.vals — Same as `calibrate.qrrvglm`. The default is a grid defined by arguments in `calibrate.rrvglm.control`.

... — Arguments that are fed into `calibrate.rrvglm.control`.

Details

Given a fitted regression CLO model, maximum likelihood calibration is theoretically easy and elegant. However, the method assumes that all responses are independent. More details and references are given in Yee (2015).

Calibration requires *grouped* or *non-sparse* data as the response. For example, if the family function is `multinomial` then one cannot usually calibrate `y0` if it is a vector of 0s except for one 1. Instead, the response vector should be from grouped data so that there are few 0s. Indeed, it is found empirically that the stereotype model (also known as a reduced-rank `multinomial` logit model) calibrates well only with grouped data, and if the response vector is all 0s except for one 1 then the MLE will probably be at `-Inf` or `+Inf`. As another example, if the family function is `poissonff` then `y0` must not be a vector of all 0s; instead, the response vector should have few 0s ideally. In general, you can use simulation to see what type of data calibrates acceptably.

Internally, this function is a simplification of `calibrate.qrrvglm` and users should look at that function for details. Good initial values are needed, and a grid is constructed to obtain these. The function `calibrate.rrvglm.control` allows the user some control over the choice of these.

Value

See `calibrate.qrrvglm`. Of course, the quadratic term in the latent variables vanishes for RR-VGLMs, so the model is simpler.

Warning

See `calibrate.qrrvglm`.

Note

See `calibrate.qrrvglm` about, e.g., calibration using real-valued responses.

Author(s)

T. W. Yee

See Also

calibrate.qrrvglm, calibrate, rrvglm, weightsvglm, optim, uniroot.

Examples

```
## Not run:   # Example 1
nona.xs.nz <- na.omit(xs.nz)  # Overkill!! (Data in VGAMdata package)
nona.xs.nz$dmd     <- with(nona.xs.nz, round(drinkmaxday))
nona.xs.nz$feethr  <- with(nona.xs.nz, round(feethour))
nona.xs.nz$sleephr <- with(nona.xs.nz, round(sleep))
nona.xs.nz$beats   <- with(nona.xs.nz, round(pulse))

p2 <- rrvglm(cbind(dmd, feethr, sleephr, beats) ~ age + smokenow +
  depressed + embarrassed + fedup + hurt + miserable +  # 11 psychological
  nofriend + moody + nervous + tense + worry + worrier, # variables
  noRRR = ~ age + smokenow, trace = FALSE, poissonff, data = nona.xs.nz,
  Rank = 2)
cp2 <- calibrate(p2, newdata = head(nona.xs.nz, 9), trace = TRUE)
cp2

two.cases <- nona.xs.nz[1:2, ]  # Another calibration example
two.cases$dmd       <- c(4, 10)
two.cases$feethr    <- c(4, 7)
two.cases$sleephr   <- c(7, 8)
two.cases$beats     <- c(62, 71)
(cp2b <- calibrate(p2, newdata = two.cases))

# Example 2
p1 <- rrvglm(cbind(dmd, feethr, sleephr, beats) ~ age + smokenow +
  depressed + embarrassed + fedup + hurt + miserable +  # 11 psychological
  nofriend + moody + nervous + tense + worry + worrier, # variables
  noRRR = ~ age + smokenow, trace = FALSE, poissonff, data = nona.xs.nz,
  Rank = 1)
(cp1c <- calibrate(p1, newdata = two.cases, lr.confint = TRUE))

## End(Not run)
```

---

calibrate.rrvglm.control *Control Function for CLO (RR-VGLM) Calibration*

Description

Algorithmic constants and parameters for running calibrate.rrvglm are set using this function.

Usage

```
calibrate.rrvglm.control(object, trace = FALSE, method.optim = "BFGS",
    gridSize = ifelse(Rank == 1, 17, 9), ...)
```

Arguments

`object`	The fitted `rrvglm` model. The user should ignore this argument.
`trace, method.optim`	Same as `calibrate.qrrvglm.control`.
`gridSize`	Same as `calibrate.qrrvglm.control`.
`...`	Avoids an error message for extraneous arguments.

Details

Most CLO users will only need to make use of `trace` and `gridSize`. These arguments should be used inside their call to `calibrate.rrvglm`, not this function directly.

Value

Similar to `calibrate.qrrvglm.control`.

See Also

`calibrate.rrvglm`, `Coef.rrvglm`.

`cao` *Fitting Constrained Additive Ordination (CAO)*

Description

A constrained additive ordination (CAO) model is fitted using the *reduced-rank vector generalized additive model* (RR-VGAM) framework.

Usage

```
cao(formula, family = stop("argument 'family' needs to be assigned"),
    data = list(),
    weights = NULL, subset = NULL, na.action = na.fail,
    etastart = NULL, mustart = NULL, coefstart = NULL,
    control = cao.control(...), offset = NULL,
    method = "cao.fit", model = FALSE, x.arg = TRUE, y.arg = TRUE,
    contrasts = NULL, constraints = NULL,
    extra = NULL, qr.arg = FALSE, smart = TRUE, ...)
```

Arguments

`formula`	a symbolic description of the model to be fit. The RHS of the formula is used to construct the latent variables, upon which the smooths are applied. All the variables in the formula are used for the construction of latent variables except for those specified by the argument noRRR, which is itself a formula. The LHS of the formula contains the response variables, which should be a matrix with each column being a response (species).

`family`	a function of class `"vglmff"` (see `vglmff-class`) describing what statistical model is to be fitted. This is called a "**VGAM** family function". See `CommonVGAMffArguments` for general information about many types of arguments found in this type of function. See `cqo` for a list of those presently implemented.
`data`	an optional data frame containing the variables in the model. By default the variables are taken from `environment(formula)`, typically the environment from which `cao` is called.
`weights`	an optional vector or matrix of (prior) weights to be used in the fitting process. For `cao`, this argument currently should not be used.
`subset`	an optional logical vector specifying a subset of observations to be used in the fitting process.
`na.action`	a function which indicates what should happen when the data contain `NA`s. The default is set by the `na.action` setting of `options`, and is `na.fail` if that is unset. The "factory-fresh" default is `na.omit`.
`etastart`	starting values for the linear predictors. It is a M-column matrix. If $M = 1$ then it may be a vector. For `cao`, this argument currently should not be used.
`mustart`	starting values for the fitted values. It can be a vector or a matrix. Some family functions do not make use of this argument. For `cao`, this argument currently should not be used.
`coefstart`	starting values for the coefficient vector. For `cao`, this argument currently should not be used.
`control`	a list of parameters for controlling the fitting process. See `cao.control` for details.
`offset`	a vector or M-column matrix of offset values. These are *a priori* known and are added to the linear predictors during fitting. For `cao`, this argument currently should not be used.
`method`	the method to be used in fitting the model. The default (and presently only) method `cao.fit` uses iteratively reweighted least squares (IRLS) within FORTRAN code called from `optim`.
`model`	a logical value indicating whether the *model frame* should be assigned in the `model` slot.
`x.arg, y.arg`	logical values indicating whether the model matrix and response vector/matrix used in the fitting process should be assigned in the `x` and `y` slots. Note the model matrix is the linear model (LM) matrix.
`contrasts`	an optional list. See the `contrasts.arg` of `model.matrix.default`.
`constraints`	an optional list of constraint matrices. For `cao`, this argument currently should not be used. The components of the list must be named with the term it corresponds to (and it must match in character format). Each constraint matrix must have M rows, and be of full-column rank. By default, constraint matrices are the M by M identity matrix unless arguments in the family function itself override these values. If `constraints` is used it must contain *all* the terms; an incomplete list is not accepted.
`extra`	an optional list with any extra information that might be needed by the family function. For `cao`, this argument currently should not be used.

`qr.arg`	For `cao`, this argument currently should not be used.
`smart`	logical value indicating whether smart prediction (`smartpred`) will be used.
`...`	further arguments passed into `cao.control`.

Details

The arguments of `cao` are a mixture of those from `vgam` and `cqo`, but with some extras in `cao.control`. Currently, not all of the arguments work properly.

CAO can be loosely be thought of as the result of fitting generalized additive models (GAMs) to several responses (e.g., species) against a very small number of latent variables. Each latent variable is a linear combination of the explanatory variables; the coefficients **C** (called C below) are called *constrained coefficients* or *canonical coefficients*, and are interpreted as weights or loadings. The **C** are estimated by maximum likelihood estimation. It is often a good idea to apply `scale` to each explanatory variable first.

For each response (e.g., species), each latent variable is smoothed by a cubic smoothing spline, thus CAO is data-driven. If each smooth were a quadratic then CAO would simplify to *constrained quadratic ordination* (CQO; formerly called *canonical Gaussian ordination* or CGO). If each smooth were linear then CAO would simplify to *constrained linear ordination* (CLO). CLO can theoretically be fitted with `cao` by specifying `df1.nl=0`, however it is more efficient to use `rrvglm`.

Currently, only `Rank=1` is implemented, and only `noRRR = ~1` models are handled.

With binomial data, the default formula is

$$logit(P[Y_s = 1]) = \eta_s = f_s(\nu), \quad s = 1, 2, \ldots, S$$

where x_2 is a vector of environmental variables, and $\nu = C^T x_2$ is a R-vector of latent variables. The η_s is an additive predictor for species s, and it models the probabilities of presence as an additive model on the logit scale. The matrix C is estimated from the data, as well as the smooth functions f_s. The argument `noRRR = ~ 1` specifies that the vector x_1, defined for RR-VGLMs and QRR-VGLMs, is simply a 1 for an intercept. Here, the intercept in the model is absorbed into the functions. A `clogloglink` link may be preferable over a `logitlink` link.

With Poisson count data, the formula is

$$\log(E[Y_s]) = \eta_s = f_s(\nu)$$

which models the mean response as an additive models on the log scale.

The fitted latent variables (site scores) are scaled to have unit variance. The concept of a tolerance is undefined for CAO models, but the optimums and maximums are defined. The generic functions `Max` and `Opt` should work for CAO objects, but note that if the maximum occurs at the boundary then `Max` will return a `NA`. Inference for CAO models is currently undeveloped.

Value

An object of class `"cao"` (this may change to `"rrvgam"` in the future). Several generic functions can be applied to the object, e.g., `Coef`, `concoef`, `lvplot`, `summary`.

Warning

CAO is very costly to compute. With version 0.7-8 it took 28 minutes on a fast machine. I hope to look at ways of speeding things up in the future.

Use `set.seed` just prior to calling `cao()` to make your results reproducible. The reason for this is finding the optimal CAO model presents a difficult optimization problem, partly because the log-likelihood function contains many local solutions. To obtain the (global) solution the user is advised to try *many* initial values. This can be done by setting `Bestof` some appropriate value (see `cao.control`). Trying many initial values becomes progressively more important as the nonlinear degrees of freedom of the smooths increase.

Note

CAO models are computationally expensive, therefore setting `trace = TRUE` is a good idea, as well as running it on a simple random sample of the data set instead.

Sometimes the IRLS algorithm does not converge within the FORTRAN code. This results in warnings being issued. In particular, if an error code of 3 is issued, then this indicates the IRLS algorithm has not converged. One possible remedy is to increase or decrease the nonlinear degrees of freedom so that the curves become more or less flexible, respectively.

Author(s)

T. W. Yee

References

Yee, T. W. (2006). Constrained additive ordination. *Ecology*, **87**, 203–213.

See Also

`cao.control`, `Coef.cao`, `cqo`, `latvar`, `Opt`, `Max`, `calibrate.qrrvglm`, `persp.cao`, `poissonff`, `binomialff`, `negbinomial`, `gamma2`, `set.seed`, `gam()` in **gam**, `trap0`.

Examples

```
## Not run:
hspider[, 1:6] <- scale(hspider[, 1:6])  # Stdzd environmental vars
set.seed(149)  # For reproducible results
ap1 <- cao(cbind(Pardlugu, Pardmont, Pardnigr, Pardpull) ~
           WaterCon + BareSand + FallTwig + CoveMoss + CoveHerb + ReflLux,
           family = poissonff, data = hspider, Rank = 1,
           df1.nl = c(Pardpull= 2.7, 2.5),
           Bestof = 7, Crow1positive = FALSE)
sort(deviance(ap1, history = TRUE))  # A history of all the iterations

Coef(ap1)
concoef(ap1)

par(mfrow = c(2, 2))
plot(ap1)  # All the curves are unimodal; some quite symmetric
```

```
par(mfrow = c(1, 1), las = 1)
index <- 1:ncol(depvar(ap1))
lvplot(ap1, lcol = index, pcol = index, y = TRUE)

trplot(ap1, label = TRUE, col = index)
abline(a = 0, b = 1, lty = 2)

trplot(ap1, label = TRUE, col = "blue", log = "xy", which.sp = c(1, 3))
abline(a = 0, b = 1, lty = 2)

persp(ap1, col = index, lwd = 2, label = TRUE)
abline(v = Opt(ap1), lty = 2, col = index)
abline(h = Max(ap1), lty = 2, col = index)

## End(Not run)
```

---

cao.control *Control Function for RR-VGAMs (CAO)*

Description

Algorithmic constants and parameters for a constrained additive ordination (CAO), by fitting a *reduced-rank vector generalized additive model* (RR-VGAM), are set using this function. This is the control function for cao.

Usage

```
cao.control(Rank = 1, all.knots = FALSE, criterion = "deviance", Cinit = NULL,
            Crow1positive = TRUE, epsilon = 1.0e-05, Etamat.colmax = 10,
            GradientFunction = FALSE, iKvector = 0.1, iShape = 0.1,
            noRRR = ~ 1, Norrr = NA,
            SmallNo = 5.0e-13, Use.Init.Poisson.QO = TRUE,
            Bestof = if (length(Cinit)) 1 else 10, maxitl = 10,
            imethod = 1, bf.epsilon = 1.0e-7, bf.maxit = 10,
            Maxit.optim = 250, optim.maxit = 20, sd.sitescores = 1.0,
            sd.Cinit = 0.02, suppress.warnings = TRUE,
            trace = TRUE, df1.nl = 2.5, df2.nl = 2.5,
            spar1 = 0, spar2 = 0, ...)
```

Arguments

Rank — The numerical rank R of the model, i.e., the number of latent variables. Currently only Rank = 1 is implemented.

all.knots — Logical indicating if all distinct points of the smoothing variables are to be used as knots. Assigning the value FALSE means fewer knots are chosen when the number of distinct points is large, meaning less computational expense. See vgam.control for details.

criterion — Convergence criterion. Currently, only one is supported: the deviance is minimized.

Cinit — Optional initial **C** matrix which may speed up convergence.

Crow1positive — Logical vector of length `Rank` (recycled if necessary): are the elements of the first row of **C** positive? For example, if `Rank` is 4, then specifying `Crow1positive = c(FALSE, TRUE)` will force **C[1,1]** and **C[1,3]** to be negative, and **C[1,2]** and **C[1,4]** to be positive.

epsilon — Positive numeric. Used to test for convergence for GLMs fitted in FORTRAN. Larger values mean a loosening of the convergence criterion.

Etamat.colmax — Positive integer, no smaller than `Rank`. Controls the amount of memory used by `.Init.Poisson.QO()`. It is the maximum number of columns allowed for the pseudo-response and its weights. In general, the larger the value, the better the initial value. Used only if `Use.Init.Poisson.QO = TRUE`.

GradientFunction — Logical. Whether `optim`'s argument `gr` is used or not, i.e., to compute gradient values. Used only if `FastAlgorithm` is `TRUE`. Currently, this argument must be set to `FALSE`.

iKvector, iShape — See `qrrvglm.control`.

noRRR — Formula giving terms that are *not* to be included in the reduced-rank regression (or formation of the latent variables). The default is to omit the intercept term from the latent variables. Currently, only `noRRR = ~ 1` is implemented.

Norrr — Defunct. Please use `noRRR`. Use of `Norrr` will become an error soon.

SmallNo — Positive numeric between `.Machine$double.eps` and `0.0001`. Used to avoid under- or over-flow in the IRLS algorithm.

Use.Init.Poisson.QO — Logical. If `TRUE` then the function `.Init.Poisson.QO` is used to obtain initial values for the canonical coefficients **C**. If `FALSE` then random numbers are used instead.

Bestof — Integer. The best of `Bestof` models fitted is returned. This argument helps guard against local solutions by (hopefully) finding the global solution from many fits. The argument works only when the function generates its own initial value for **C**, i.e., when **C** are *not* passed in as initial values. The default is only a convenient minimal number and users are urged to increase this value.

maxitl — Positive integer. Maximum number of Newton-Raphson/Fisher-scoring/local-scoring iterations allowed.

imethod — See `qrrvglm.control`.

bf.epsilon — Positive numeric. Tolerance used by the modified vector backfitting algorithm for testing convergence.

bf.maxit — Positive integer. Number of backfitting iterations allowed in the compiled code.

Maxit.optim — Positive integer. Number of iterations given to the function `optim` at each of the `optim.maxit` iterations.

optim.maxit — Positive integer. Number of times `optim` is invoked.

sd.sitescores
: Numeric. Standard deviation of the initial values of the site scores, which are generated from a normal distribution. Used when `Use.Init.Poisson.QO` is `FALSE`.

sd.Cinit
: Standard deviation of the initial values for the elements of **C**. These are normally distributed with mean zero. This argument is used only if `Use.Init.Poisson.QO = FALSE`.

suppress.warnings
: Logical. Suppress warnings?

trace
: Logical indicating if output should be produced for each iteration. Having the value `TRUE` is a good idea for large data sets.

df1.nl, df2.nl
: Numeric and non-negative, recycled to length S. Nonlinear degrees of freedom for smooths of the first and second latent variables. A value of 0 means the smooth is linear. Roughly, a value between 1.0 and 2.0 often has the approximate flexibility of a quadratic. The user should not assign too large a value to this argument, e.g., the value 4.0 is probably too high. The argument `df1.nl` is ignored if `spar1` is assigned a positive value or values. Ditto for `df2.nl`.

spar1, spar2
: Numeric and non-negative, recycled to length S. Smoothing parameters of the smooths of the first and second latent variables. The larger the value, the more smooth (less wiggly) the fitted curves. These arguments are an alternative to specifying `df1.nl` and `df2.nl`. A value 0 (the default) for `spar1` means that `df1.nl` is used. Ditto for `spar2`. The values are on a scaled version of the latent variables. See Green and Silverman (1994) for more information.

...
: Ignored at present.

Details

Many of these arguments are identical to `qrrvglm.control`. Here, R is the `Rank`, M is the number of additive predictors, and S is the number of responses (species). Thus $M = S$ for binomial and Poisson responses, and $M = 2S$ for the negative binomial and 2-parameter gamma distributions.

Allowing the smooths too much flexibility means the CAO optimization problem becomes more difficult to solve. This is because the number of local solutions increases as the nonlinearity of the smooths increases. In situations of high nonlinearity, many initial values should be used, so that `Bestof` should be assigned a larger value. In general, there should be a reasonable value of `df1.nl` somewhere between 0 and about 3 for most data sets.

Value

A list with the components corresponding to its arguments, after some basic error checking.

Note

The argument `df1.nl` can be inputted in the format `c(spp1 = 2, spp2 = 3, 2.5)`, say, meaning the default value is 2.5, but two species have alternative values.

If `spar1 = 0` and `df1.nl = 0` then this represents fitting linear functions (CLO). Currently, this is handled in the awkward manner of setting `df1.nl` to be a small positive value, so that the smooth is almost linear but not quite. A proper fix to this special case should done in the short future.

Author(s)

T. W. Yee

References

Yee, T. W. (2006). Constrained additive ordination. *Ecology*, **87**, 203–213.

Green, P. J. and Silverman, B. W. (1994). *Nonparametric Regression and Generalized Linear Models: A Roughness Penalty Approach*, London: Chapman & Hall.

See Also

cao.

Examples

```
## Not run:
hspider[,1:6] <- scale(hspider[,1:6])  # Standardized environmental vars
set.seed(123)
ap1 <- cao(cbind(Pardlugu, Pardmont, Pardnigr, Pardpull, Zoraspin) ~
           WaterCon + BareSand + FallTwig +
           CoveMoss + CoveHerb + ReflLux,
           family = poissonff, data = hspider,
           df1.nl = c(Zoraspin = 2.3, 2.1),
           Bestof = 10, Crow1positive = FALSE)
sort(deviance(ap1, history = TRUE))  # A history of all the iterations

Coef(ap1)

par(mfrow = c(2, 3))  # All or most of the curves are unimodal; some are
plot(ap1, lcol = "blue")  # quite symmetric. Hence a CQO model should be ok

par(mfrow = c(1, 1), las = 1)
index <- 1:ncol(depvar(ap1))  # lvplot is jagged because only 28 sites
lvplot(ap1, lcol = index, pcol = index, y = TRUE)

trplot(ap1, label = TRUE, col = index)
abline(a = 0, b = 1, lty = 2)

persp(ap1, label = TRUE, col = 1:4)

## End(Not run)
```

Card *Cardioid Distribution*

Description

Density, distribution function, quantile function and random generation for the cardioid distribution.

Usage

```
dcard(x, mu, rho, log = FALSE)
pcard(q, mu, rho, lower.tail = TRUE, log.p = FALSE)
qcard(p, mu, rho, tolerance = 1e-07, maxits = 500,
      lower.tail = TRUE, log.p = FALSE)
rcard(n, mu, rho, ...)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. Same as in `runif`.
`mu, rho`	See `cardioid` for more information.
`tolerance, maxits, ...`	The first two are control parameters for the algorithm used to solve for the roots of a nonlinear system of equations; `tolerance` controls for the accuracy and `maxits` is the maximum number of iterations. `rcard` calls `qcard` so the `...` can be used to vary the two arguments.
`log`	Logical. If `log = TRUE` then the logarithm of the density is returned.
`lower.tail, log.p`	Same meaning as in `pnorm` or `qnorm`.

Details

See `cardioid`, the **VGAM** family function for estimating the two parameters by maximum likelihood estimation, for the formula of the probability density function and other details.

Value

`dcard` gives the density, `pcard` gives the distribution function, `qcard` gives the quantile function, and `rcard` generates random deviates.

Note

Convergence problems might occur with `rcard`.

Author(s)

Thomas W. Yee and Kai Huang

See Also

`cardioid`.

Examples

```
## Not run:
mu <- 4; rho <- 0.4; x <- seq(0, 2*pi, len = 501)
plot(x, dcard(x, mu, rho), type = "l", las = 1, ylim = c(0, 1),
     ylab = paste("[dp]card(mu=", mu, ", rho=", rho, ")"),
     main = "Blue is density, orange is the CDF", col = "blue",
     sub = "Purple lines are the 10,20,...,90 percentiles")
lines(x, pcard(x, mu, rho), col = "orange")

probs <- seq(0.1, 0.9, by = 0.1)
Q <- qcard(probs, mu, rho)
lines(Q, dcard(Q, mu, rho), col = "purple", lty = 3, type = "h")
lines(Q, pcard(Q, mu, rho), col = "purple", lty = 3, type = "h")
abline(h = c(0,probs, 1), v = c(0, 2*pi), col = "purple", lty = 3)
max(abs(pcard(Q, mu, rho) - probs))  # Should be 0

## End(Not run)
```

`cardioid` *Cardioid Distribution Family Function*

Description

Estimates the two parameters of the cardioid distribution by maximum likelihood estimation.

Usage

```
cardioid(lmu = extlogitlink(min = 0, max = 2*pi),
          lrho = extlogitlink(min = -0.5, max = 0.5),
          imu = NULL, irho = 0.3, nsimEIM = 100, zero = NULL)
```

Arguments

`lmu, lrho`	Parameter link functions applied to the μ and ρ parameters, respectively. See `Links` for more choices.
`imu, irho`	Initial values. A `NULL` means an initial value is chosen internally. See `CommonVGAMffArguments` for more information.
`nsimEIM, zero`	See `CommonVGAMffArguments` for more information.

Details

The two-parameter cardioid distribution has a density that can be written as

$$f(y; \mu, \rho) = \frac{1}{2\pi} (1 + 2\,\rho \cos(y - \mu))$$

where $0 < y < 2\pi$, $0 < \mu < 2\pi$, and $-0.5 < \rho < 0.5$ is the concentration parameter. The default link functions enforce the range constraints of the parameters.

For positive ρ the distribution is unimodal and symmetric about μ. The mean of Y (which make up the fitted values) is $\pi + (\rho/\pi)((2\pi - \mu)\sin(2\pi - \mu) + \cos(2\pi - \mu) - \mu\sin(\mu) - \cos(\mu))$.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, rrvglm and vgam.

Warning

Numerically, this distribution can be difficult to fit because of a log-likelihood having multiple maximums. The user is therefore encouraged to try different starting values, i.e., make use of imu and irho.

Note

Fisher scoring using simulation is used.

Author(s)

T. W. Yee

References

Jammalamadaka, S. R. and SenGupta, A. (2001). *Topics in Circular Statistics*, Singapore: World Scientific.

See Also

rcard, extlogitlink, vonmises.

CircStats and **circular** currently have a lot more R functions for circular data than the **VGAM** package.

Examples

```
## Not run:
cdata <- data.frame(y = rcard(n = 1000, mu = 4, rho = 0.45))
fit <- vglm(y ~ 1, cardioid, data = cdata, trace = TRUE)
coef(fit, matrix=TRUE)
Coef(fit)
c(with(cdata, mean(y)), head(fitted(fit), 1))
summary(fit)

## End(Not run)
```

`cauchitlink` *Cauchit Link Function*

Description

Computes the cauchit (tangent) link transformation, including its inverse and the first two derivatives.

Usage

```
cauchitlink(theta, bvalue = .Machine$double.eps,
            inverse = FALSE, deriv = 0, short = TRUE, tag = FALSE)
```

Arguments

`theta`	Numeric or character. See below for further details.
`bvalue`	See `Links`.
`inverse, deriv, short, tag`	Details at `Links`.

Details

This link function is an alternative link function for parameters that lie in the unit interval. This type of link bears the same relation to the Cauchy distribution as the probit link bears to the Gaussian. One characteristic of this link function is that the tail is heavier relative to the other links (see examples below).

Numerical values of `theta` close to 0 or 1 or out of range result in `Inf`, `-Inf`, `NA` or `NaN`.

Value

For `deriv = 0`, the tangent of `theta`, i.e., `tan(pi * (theta-0.5))` when `inverse = FALSE`, and if `inverse = TRUE` then `0.5 + atan(theta)/pi`.

For `deriv = 1`, then the function returns *d* `eta` / *d* `theta` as a function of `theta` if `inverse = FALSE`, else if `inverse = TRUE` then it returns the reciprocal.

Note

Numerical instability may occur when `theta` is close to 1 or 0. One way of overcoming this is to use `bvalue`.

As mentioned above, in terms of the threshold approach with cumulative probabilities for an ordinal response this link function corresponds to the Cauchy distribution (see `cauchy1`).

Author(s)

Thomas W. Yee

References

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*, 2nd ed. London: Chapman & Hall.

See Also

logitlink, probitlink, clogloglink, loglink, cauchy, cauchy1, Cauchy.

Examples

```
p <- seq(0.01, 0.99, by = 0.01)
cauchitlink(p)
max(abs(cauchitlink(cauchitlink(p), inverse = TRUE) - p))  # Should be 0

p <- c(seq(-0.02, 0.02, by=0.01), seq(0.97, 1.02, by = 0.01))
cauchitlink(p)  # Has no NAs

## Not run:
par(mfrow = c(2, 2), lwd = (mylwd <- 2))
y <- seq(-4, 4, length = 100)
p <- seq(0.01, 0.99, by = 0.01)

for (d in 0:1) {
  matplot(p, cbind(logitlink(p, deriv = d), probitlink(p, deriv = d)),
          type = "n", col = "purple", ylab = "transformation",
          las = 1, main = if (d == 0) "Some probability link functions"
          else "First derivative")
  lines(p,   logitlink(p, deriv = d), col = "limegreen")
  lines(p,  probitlink(p, deriv = d), col = "purple")
  lines(p, clogloglink(p, deriv = d), col = "chocolate")
  lines(p, cauchitlink(p, deriv = d), col = "tan")
  if (d == 0) {
    abline(v = 0.5, h = 0, lty = "dashed")
    legend(0, 4.5, c("logitlink", "probitlink", "clogloglink",
           "cauchitlink"), lwd = mylwd,
           col = c("limegreen", "purple", "chocolate", "tan"))
  } else
    abline(v = 0.5, lty = "dashed")
}

for (d in 0) {
  matplot(y, cbind( logitlink(y, deriv = d, inverse = TRUE),
                   probitlink(y, deriv = d, inverse = TRUE)),
          type  = "n", col = "purple", xlab = "transformation", ylab = "p",
          main = if (d == 0) "Some inverse probability link functions"
          else "First derivative", las=1)
  lines(y,   logitlink(y, deriv = d, inverse = TRUE), col = "limegreen")
  lines(y,  probitlink(y, deriv = d, inverse = TRUE), col = "purple")
  lines(y, clogloglink(y, deriv = d, inverse = TRUE), col = "chocolate")
  lines(y, cauchitlink(y, deriv = d, inverse = TRUE), col = "tan")
  if (d == 0) {
      abline(h = 0.5, v = 0, lty = "dashed")
```

```
    legend(-4, 1, c("logitlink", "probitlink", "clogloglink",
           "cauchitlink"), lwd = mylwd,
           col = c("limegreen", "purple", "chocolate", "tan"))
  }
}
par(lwd = 1)

## End(Not run)
```

---

cauchy *Cauchy Distribution Family Function*

Description

Estimates either the location parameter or both the location and scale parameters of the Cauchy distribution by maximum likelihood estimation.

Usage

```
cauchy(llocation = "identitylink", lscale = "loglink",
       imethod = 1, ilocation = NULL, iscale = NULL,
       gprobs.y = ppoints(19), gscale.mux = exp(-3:3), zero = "scale")
cauchy1(scale.arg = 1, llocation = "identitylink", ilocation = NULL,
        imethod = 1, gprobs.y = ppoints(19), zero = NULL)
```

Arguments

`llocation, lscale`
: Parameter link functions for the location parameter a and the scale parameter b. See `Links` for more choices.

`ilocation, iscale`
: Optional initial value for a and b. By default, an initial value is chosen internally for each.

`imethod`
: Integer, either 1 or 2 or 3. Initial method, three algorithms are implemented. The user should try all possible values to help avoid converging to a local solution. Also, choose the another value if convergence fails, or use `ilocation` and/or `iscale`.

`gprobs.y, gscale.mux, zero`
: See `CommonVGAMffArguments` for information.

`scale.arg`
: Known (positive) scale parameter, called b below.

Details

The Cauchy distribution has density function

$$f(y; a, b) = \left\{\pi b[1 + ((y - a)/b)^2]\right\}^{-1}$$

where y and a are real and finite, and $b > 0$. The distribution is symmetric about a and has a heavy tail. Its median and mode are a, but the mean does not exist. The fitted values are the estimates of a. Fisher scoring is used.

If the scale parameter is known (cauchy1) then there may be multiple local maximum likelihood solutions for the location parameter. However, if both location and scale parameters are to be estimated (cauchy) then there is a unique maximum likelihood solution provided $n > 2$ and less than half the data are located at any one point.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Warning

It is well-known that the Cauchy distribution may have local maximums in its likelihood function; make full use of imethod, ilocation, iscale etc.

Note

Good initial values are needed. By default cauchy searches for a starting value for a and b on a 2-D grid. Likewise, by default, cauchy1 searches for a starting value for a on a 1-D grid. If convergence to the global maximum is not acheieved then it also pays to select a wide range of initial values via the ilocation and/or iscale and/or imethod arguments.

Author(s)

T. W. Yee

References

Forbes, C., Evans, M., Hastings, N. and Peacock, B. (2011). *Statistical Distributions*, Hoboken, NJ, USA: John Wiley and Sons, Fourth edition.

Barnett, V. D. (1966). Evaluation of the maximum-likehood estimator where the likelihood equation has multiple roots. *Biometrika*, **53**, 151–165.

Copas, J. B. (1975). On the unimodality of the likelihood for the Cauchy distribution. *Biometrika*, **62**, 701–704.

Efron, B. and Hinkley, D. V. (1978). Assessing the accuracy of the maximum likelihood estimator: Observed versus expected Fisher information. *Biometrika*, **65**, 457–481.

See Also

Cauchy, cauchit, studentt, simulate.vlm.

Examples

```
# Both location and scale parameters unknown
set.seed(123)
cdata <- data.frame(x2 = runif(nn <- 1000))
cdata <- transform(cdata, loc = exp(1 + 0.5 * x2), scale = exp(1))
cdata <- transform(cdata, y2 = rcauchy(nn, loc, scale))
fit2 <- vglm(y2 ~ x2, cauchy(lloc = "loglink"), data = cdata)
coef(fit2, matrix = TRUE)
head(fitted(fit2))  # Location estimates
summary(fit2)

# Location parameter unknown
cdata <- transform(cdata, scale1 = 0.4)
cdata <- transform(cdata, y1 = rcauchy(nn, loc, scale1))
fit1 <- vglm(y1 ~ x2, cauchy1(scale = 0.4), data = cdata, trace = TRUE)
coef(fit1, matrix = TRUE)
```

`cdf.lmscreg` *Cumulative Distribution Function for LMS Quantile Regression*

Description

Computes the cumulative distribution function (CDF) for observations, based on a LMS quantile regression.

Usage

```
cdf.lmscreg(object, newdata = NULL, ...)
```

Arguments

`object`	A **VGAM** quantile regression model, i.e., an object produced by modelling functions such as `vglm` and `vgam` with a family function beginning with `"lms."`.
`newdata`	Data frame where the predictions are to be made. If missing, the original data is used.
`...`	Parameters which are passed into functions such as `cdf.lms.yjn`.

Details

The CDFs returned here are values lying in [0,1] giving the relative probabilities associated with the quantiles `newdata`. For example, a value near 0.75 means it is close to the upper quartile of the distribution.

Value

A vector of CDF values lying in [0,1].

Note

The data are treated like quantiles, and the percentiles are returned. The opposite is performed by qtplot.lmscreg.

The CDF values of the model have been placed in @post$cdf when the model was fitted.

Author(s)

Thomas W. Yee

References

Yee, T. W. (2004). Quantile regression via vector generalized additive models. *Statistics in Medicine*, **23**, 2295–2315.

See Also

deplot.lmscreg, qtplot.lmscreg, lms.bcn, lms.bcg, lms.yjn, CommonVGAMffArguments.

Examples

```
fit <- vgam(BMI ~ s(age, df=c(4, 2)), lms.bcn(zero = 1), data = bmi.nz)
head(fit@post$cdf)
head(cdf(fit))  # Same
head(depvar(fit))
head(fitted(fit))

cdf(fit, data.frame(age = c(31.5, 39), BMI = c(28.4, 24)))
```

---

cens.gumbel *Censored Gumbel Distribution*

Description

Maximum likelihood estimation of the 2-parameter Gumbel distribution when there are censored observations. A matrix response is not allowed.

Usage

```
cens.gumbel(llocation = "identitylink", lscale = "loglink",
            iscale = NULL, mean = TRUE, percentiles = NULL,
            zero = "scale")
```

Arguments

`llocation, lscale`
: Character. Parameter link functions for the location and (positive) $scale$ parameters. See `Links` for more choices.

`iscale`
: Numeric and positive. Initial value for $scale$. Recycled to the appropriate length. In general, a larger value is better than a smaller value. The default is to choose the value internally.

`mean`
: Logical. Return the mean? If `TRUE` then the mean is returned, otherwise percentiles given by the `percentiles` argument.

`percentiles`
: Numeric with values between 0 and 100. If `mean=FALSE` then the fitted values are percentiles which must be specified by this argument.

`zero`
: An integer-valued vector specifying which linear/additive predictors are modelled as intercepts only. The value (possibly values) must be from the set {1,2} corresponding respectively to $location$ and $scale$. If `zero=NULL` then all linear/additive predictors are modelled as a linear combination of the explanatory variables. The default is to fit the shape parameter as an intercept only. See `CommonVGAMffArguments` for more information.

Details

This **VGAM** family function is like `gumbel` but handles observations that are left-censored (so that the true value would be less than the observed value) else right-censored (so that the true value would be greater than the observed value). To indicate which type of censoring, input `extra = list(leftcensored = vec1, rightcensored = vec2)` where `vec1` and `vec2` are logical vectors the same length as the response. If the two components of this list are missing then the logical values are taken to be `FALSE`. The fitted object has these two components stored in the `extra` slot.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Warning

Numerical problems may occur if the amount of censoring is excessive.

Note

See `gumbel` for details about the Gumbel distribution. The initial values are based on assuming all uncensored observations, therefore could be improved upon.

Author(s)

T. W. Yee

References

Coles, S. (2001). *An Introduction to Statistical Modeling of Extreme Values*. London: Springer-Verlag.

See Also

gumbel, gumbelff, rgumbel, guplot, gev, venice.

Examples

```
# Example 1
ystar <- venice[["r1"]]  # Use the first order statistic as the response
nn <- length(ystar)
L <- runif(nn, 100, 104)  # Lower censoring points
U <- runif(nn, 130, 135)  # Upper censoring points
y <- pmax(L, ystar)  # Left  censored
y <- pmin(U, y)      # Right censored
extra <- list(leftcensored = ystar < L, rightcensored = ystar > U)
fit <- vglm(y ~ scale(year), data = venice, trace = TRUE, extra = extra,
            fam = cens.gumbel(mean = FALSE, perc = c(5, 25, 50, 75, 95)))
coef(fit, matrix = TRUE)
head(fitted(fit))
fit@extra

# Example 2: simulated data
nn <- 1000
ystar <- rgumbel(nn, loc = 1, scale = exp(0.5))  # The uncensored data
L <- runif(nn, -1, 1)  # Lower censoring points
U <- runif(nn,  2, 5)  # Upper censoring points
y <- pmax(L, ystar)  # Left  censored
y <- pmin(U, y)      # Right censored
## Not run: par(mfrow = c(1, 2)); hist(ystar); hist(y);
extra <- list(leftcensored = ystar < L, rightcensored = ystar > U)
fit <- vglm(y ~ 1, trace = TRUE, extra = extra, fam = cens.gumbel)
coef(fit, matrix = TRUE)
```

---

cens.normal *Censored Normal Distribution*

Description

Maximum likelihood estimation for the normal distribution with left and right censoring.

Usage

```
cens.normal(lmu = "identitylink", lsd = "loglink", imethod = 1,
             zero = "sd")
```

Arguments

lmu, lsd	Parameter link functions applied to the mean and standard deviation parameters. See Links for more choices. The standard deviation is a positive quantity, therefore a log link is the default.

`imethod` Initialization method. Either 1 or 2, this specifies two methods for obtaining initial values for the parameters.

`zero` A vector, e.g., containing the value 1 or 2; if so, the mean or standard deviation respectively are modelled as an intercept only. Setting `zero = NULL` means both linear/additive predictors are modelled as functions of the explanatory variables. See `CommonVGAMffArguments` for more information.

Details

This function is like `uninormal` but handles observations that are left-censored (so that the true value would be less than the observed value) else right-censored (so that the true value would be greater than the observed value). To indicate which type of censoring, input `extra = list(leftcensored = vec1, rightcensored = vec2)` where `vec1` and `vec2` are logical vectors the same length as the response. If the two components of this list are missing then the logical values are taken to be `FALSE`. The fitted object has these two components stored in the `extra` slot.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Note

This function, which is an alternative to `tobit`, cannot handle a matrix response and uses different working weights. If there are no censored observations then `uninormal` is recommended instead.

Author(s)

T. W. Yee

See Also

`tobit`, `uninormal`, `double.cens.normal`.

Examples

```
## Not run:
cdata <- data.frame(x2 = runif(nn <- 1000))  # ystar are true values
cdata <- transform(cdata, ystar = rnorm(nn, m = 100 + 15 * x2, sd = exp(3)))
with(cdata, hist(ystar))
cdata <- transform(cdata, L = runif(nn,  80,  90),  # Lower censoring points
                          U = runif(nn, 130, 140))  # Upper censoring points
cdata <- transform(cdata, y = pmax(L, ystar))  # Left  censored
cdata <- transform(cdata, y = pmin(U, y))      # Right censored
with(cdata, hist(y))
Extra <- list(leftcensored  = with(cdata, ystar < L),
              rightcensored = with(cdata, ystar > U))
fit1 <- vglm(y ~ x2, cens.normal, data = cdata, crit = "c", extra = Extra)
fit2 <- vglm(y ~ x2, tobit(Lower = with(cdata, L), Upper = with(cdata, U)),
            data = cdata, crit = "c", trace = TRUE)
coef(fit1, matrix = TRUE)
```

```
max(abs(coef(fit1, matrix = TRUE) -
        coef(fit2, matrix = TRUE)))  # Should be 0
names(fit1@extra)

## End(Not run)
```

`cens.poisson` *Censored Poisson Family Function*

Description

Family function for a censored Poisson response.

Usage

```
cens.poisson(link = "loglink", imu = NULL,
             biglambda = 10, smallno = 1e-10)
```

Arguments

`link`	Link function applied to the mean; see `Links` for more choices.
`imu`	Optional initial value; see `CommonVGAMffArguments` for more information.
`biglambda, smallno`	Used to help robustify the code when `lambda` is very large and the `ppois` value is so close to 0 that the first derivative is computed to be a `NA` or `NaN`. When this occurs `mills.ratio` is called.

Details

Often a table of Poisson counts has an entry $J+$ meaning $\geq J$. This family function is similar to `poissonff` but handles such censored data. The input requires `SurvS4`. Only a univariate response is allowed. The Newton-Raphson algorithm is used.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Warning

As the response is discrete, care is required with `Surv`, especially with `"interval"` censored data because of the `(start, end]` format. See the examples below. The examples have `y < L` as left censored and `y >= U` (formatted as `U+`) as right censored observations, therefore `L <= y < U` is for uncensored and/or interval censored observations. Consequently the input must be tweaked to conform to the `(start, end]` format.

A bit of attention has been directed to try robustify the code when `lambda` is very large, however this currently works for left and right censored data only, not interval censored data. Sometime the fix involves an approximation, hence it is a good idea to set `trace = TRUE`.

Note

The function poissonff should be used when there are no censored observations. Also, NAs are not permitted with SurvS4, nor is type = "counting".

Author(s)

Thomas W. Yee

References

See **survival** for background.

See Also

SurvS4, poissonff, Links, mills.ratio.

Examples

```
# Example 1: right censored data
set.seed(123); U <- 20
cdata <- data.frame(y = rpois(N <- 100, exp(3)))
cdata <- transform(cdata, cy = pmin(U, y),
                          rcensored = (y >= U))
cdata <- transform(cdata, status = ifelse(rcensored, 0, 1))
with(cdata, table(cy))
with(cdata, table(rcensored))
with(cdata, table(print(SurvS4(cy, status))))  # Check; U+ means >= U
fit <- vglm(SurvS4(cy, status) ~ 1, cens.poisson, data = cdata,
            trace = TRUE)
coef(fit, matrix = TRUE)
table(print(depvar(fit)))  # Another check; U+ means >= U

# Example 2: left censored data
L <- 15
cdata <- transform(cdata,
               cY = pmax(L, y),
               lcensored = y <  L)  # Note y < L, not cY == L or y <= L
cdata <- transform(cdata, status = ifelse(lcensored, 0, 1))
with(cdata, table(cY))
with(cdata, table(lcensored))
with(cdata, table(print(SurvS4(cY, status, type = "left"))))  # Check
fit <- vglm(SurvS4(cY, status, type = "left") ~ 1, cens.poisson,
            data = cdata, trace = TRUE)
coef(fit, matrix = TRUE)

# Example 3: interval censored data
cdata <- transform(cdata, Lvec = rep(L, len = N),
                          Uvec = rep(U, len = N))
cdata <-
  transform(cdata,
        icensored = Lvec <= y & y < Uvec)  # Not lcensored or rcensored
with(cdata, table(icensored))
```

```
cdata <- transform(cdata, status = rep(3, N))  # 3 == interval censored
cdata <- transform(cdata,
         status = ifelse(rcensored, 0, status))  # 0 means right censored
cdata <- transform(cdata,
         status = ifelse(lcensored, 2, status))  # 2 means left  censored
# Have to adjust Lvec and Uvec because of the (start, end] format:
cdata$Lvec[with(cdata,icensored)] <- cdata$Lvec[with(cdata,icensored)]-1
cdata$Uvec[with(cdata,icensored)] <- cdata$Uvec[with(cdata,icensored)]-1
# Unchanged:
cdata$Lvec[with(cdata, lcensored)] <- cdata$Lvec[with(cdata, lcensored)]
cdata$Lvec[with(cdata, rcensored)] <- cdata$Uvec[with(cdata, rcensored)]
with(cdata,  # Check
 table(ii <- print(SurvS4(Lvec, Uvec, status, type = "interval"))))
fit <- vglm(SurvS4(Lvec, Uvec, status, type = "interval") ~ 1,
            cens.poisson, data = cdata, trace = TRUE)
coef(fit, matrix = TRUE)
table(print(depvar(fit)))  # Another check

# Example 4: Add in some uncensored observations
index <- (1:N)[with(cdata, icensored)]
index <- head(index, 4)
cdata$status[index] <- 1  # actual or uncensored value
cdata$Lvec[index] <- cdata$y[index]
with(cdata, table(ii <- print(SurvS4(Lvec, Uvec, status,
                                     type = "interval"))))  # Check
fit <- vglm(SurvS4(Lvec, Uvec, status, type = "interval") ~ 1,
            cens.poisson, data = cdata, trace = TRUE, crit = "c")
coef(fit, matrix = TRUE)
table(print(depvar(fit)))  # Another check
```

`cfibrosis` *Cystic Fibrosis Data*

Description

This data frame concerns families data and cystic fibrosis.

Usage

```
data(cfibrosis)
```

Format

A data frame with 24 rows on the following 4 variables.

siblings, affected, ascertained, families Over ascertained families, the kth ascertained family has s_k siblings of whom r_k are affected and a_k are ascertained.

Details

The data set allows a classical segregation analysis to be peformed. In particular, to test Mendelian segregation ratios in nuclear family data. The likelihood has similarities with `seq2binomial`.

Source

The data is originally from Crow (1965) and appears as Table 2.3 of Lange (2002).

Crow, J. F. (1965) Problems of ascertainment in the analysis of family data. Epidemiology and Genetics of Chronic Disease. Public Health Service Publication 1163, Neel J. V., Shaw M. W., Schull W. J., editors, Department of Health, Education, and Welfare, Washington, DC, USA.

Lange, K. (2002) Mathematical and Statistical Methods for Genetic Analysis. Second Edition. Springer-Verlag: New York, USA.

Examples

```
cfibrosis
summary(cfibrosis)
```

`cgo` *Redirects the user to cqo*

Description

Redirects the user to the function `cqo`.

Usage

```
cgo(...)
```

Arguments

`...`	Ignored.

Details

The former function `cgo` has been renamed `cqo` because CGO (for *canonical Gaussian ordination*) is a confusing and inaccurate name. CQO (for *constrained quadratic ordination*) is better. This new nomenclature described in Yee (2006).

Value

Nothing is returned; an error message is issued.

Warning

The code, therefore, in Yee (2004) will not run without changing the `"g"` to a `"q"`.

Author(s)

Thomas W. Yee

References

Yee, T. W. (2004). A new technique for maximum-likelihood canonical Gaussian ordination. *Ecological Monographs*, **74**, 685–701.

Yee, T. W. (2006). Constrained additive ordination. *Ecology*, **87**, 203–213.

See Also

cqo.

Examples

```
## Not run:
cgo()

## End(Not run)
```

---

chest.nz *Chest Pain in NZ Adults Data*

Description

Presence/absence of chest pain in 10186 New Zealand adults.

Usage

```
data(chest.nz)
```

Format

A data frame with 73 rows and the following 5 variables.

age a numeric vector; age (years).

nolnor a numeric vector of counts; no pain on LHS or RHS.

nolr a numeric vector of counts; no pain on LHS but pain on RHS.

lnor a numeric vector of counts; no pain on RHS but pain on LHS.

lr a numeric vector of counts; pain on LHS and RHS of chest.

Details

Each adult was asked their age and whether they experienced any pain or discomfort in their chest over the last six months. If yes, they indicated whether it was on their LHS and/or RHS of their chest.

Source

MacMahon, S., Norton, R., Jackson, R., Mackie, M. J., Cheng, A., Vander Hoorn, S., Milne, A., McCulloch, A. (1995) Fletcher Challenge-University of Auckland Heart & Health Study: design and baseline findings. *New Zealand Medical Journal*, **108**, 499–502.

Examples

```
## Not run:
fit <- vgam(cbind(nolnor, nolr, lnor, lr) ~ s(age, c(4, 3)),
            binom2.or(exchan = TRUE, zero = NULL), data = chest.nz)
coef(fit, matrix = TRUE)

## End(Not run)
## Not run:  plot(fit, which.cf = 2, se = TRUE)
```

---

chinese.nz *Chinese Population in New Zealand 1867–2001 Data*

Description

The Chinese population in New Zealand from 1867 to 2001, along with the whole of the New Zealand population.

Usage

```
data(chinese.nz)
```

Format

A data frame with 27 observations on the following 4 variables.

`year` Year.

`male` Number of Chinese males.

`female` Number of Chinese females.

`nz` Total number in the New Zealand population.

Details

Historically, there was a large exodus of Chinese from the Guangdong region starting in the mid-1800s to the gold fields of South Island of New Zealand, California (a region near Mexico), and southern Australia, etc. Discrimination then meant that only men were allowed entry, to hinder permanent settlement. In the case of New Zealand, the government relaxed its immigration laws after WWII to allow wives of Chinese already in NZ to join them because China had been among the Allied powers. Gradual relaxation in the immigration and an influx during the 1980s meant the Chinese population became increasingly demographically normal over time.

The NZ total for the years 1867 and 1871 exclude the Maori population. Three modifications have been made to the female column to make the data internally consistent with the original table.

References

Page 6 of *Aliens At My Table: Asians as New Zealanders See Them* by M. Ip and N. Murphy, (2005). Penguin Books. Auckland, New Zealand.

Examples

```
## Not run:  par(mfrow = c(1, 2))
plot(female / (male + female) ~ year, chinese.nz, type = "b",
     ylab = "Proportion", col = "blue", las = 1,
     cex = 0.015 * sqrt(male + female),
#    cex = 0.10 * sqrt((male + female)^1.5 / sqrt(female) / sqrt(male)),
     main = "Proportion of NZ Chinese that are female")
abline(h = 0.5, lty = "dashed", col = "gray")

fit1.cnz <- vglm(cbind(female, male) ~ year,             binomialff,
                 data = chinese.nz)
fit2.cnz <- vglm(cbind(female, male) ~ sm.poly(year, 2), binomialff,
                 data = chinese.nz)
fit4.cnz <- vglm(cbind(female, male) ~   sm.bs(year, 5), binomialff,
                 data = chinese.nz)

lines(fitted(fit1.cnz) ~ year, chinese.nz, col = "purple", lty = 1)
lines(fitted(fit2.cnz) ~ year, chinese.nz, col = "green", lty = 2)
lines(fitted(fit4.cnz) ~ year, chinese.nz, col = "orange", lwd = 2, lty = 1)
legend("bottomright", col = c("purple", "green", "orange"),
       lty = c(1, 2, 1), leg = c("linear", "quadratic", "B-spline"))

plot(100*(male+female)/nz ~ year, chinese.nz, type = "b", ylab = "Percent",
     ylim = c(0, max(100*(male+female)/nz)), col = "blue", las = 1,
     main = "Percent of NZers that are Chinese")
abline(h = 0, lty = "dashed", col = "gray")
## End(Not run)
```

---

chisq — *Chi-squared and Chi Distributions*

Description

Maximum likelihood estimation of the degrees of freedom for a chi-squared distribution. Also fits the chi distribution.

Usage

```
chisq(link = "loglink", zero = NULL, squared = TRUE)
```

Arguments

`link, zero`	See `CommonVGAMffArguments` for information.
`squared`	Logical. Set `FALSE` for the chi distribution.

Details

The degrees of freedom is treated as a real parameter to be estimated and not as an integer. Being positive, a log link is used by default. Fisher scoring is used.

If a random variable has a chi-squared distribution then the square root of the random variable has a chi distribution. For both distributions, the fitted value is the mean.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Note

Multiple responses are permitted. There may be convergence problems if the degrees of freedom is very large or close to zero.

Author(s)

T. W. Yee

References

Forbes, C., Evans, M., Hastings, N. and Peacock, B. (2011). *Statistical Distributions*, Hoboken, NJ, USA: John Wiley and Sons, Fourth edition.

See Also

Chisquare. uninormal.

Examples

```
cdata <- data.frame(x2 = runif(nn <- 1000))
cdata <- transform(cdata, y1 = rchisq(nn, df = exp(1 - 1 * x2)),
                          y2 = rchisq(nn, df = exp(2 - 2 * x2)))
fit <- vglm(cbind(y1, y2) ~ x2, chisq, data = cdata, trace = TRUE)
coef(fit, matrix = TRUE)
```

---

clo — *Redirects the User to rrvglm()*

Description

Redirects the user to the function rrvglm.

Usage

```
clo(...)
```

Arguments

... Ignored.

Details

CLO stands for *constrained linear ordination*, and is fitted with a statistical class of models called *reduced-rank vector generalized linear models* (RR-VGLMs). It allows for generalized reduced-rank regression in that response types such as Poisson counts and presence/absence data can be handled.

Currently in the **VGAM** package, `rrvglm` is used to fit RR-VGLMs. However, the Author's opinion is that linear responses to a latent variable (composite environmental gradient) is not as common as unimodal responses, therefore `cqo` is often more appropriate.

The new CLO/CQO/CAO nomenclature described in Yee (2006).

Value

Nothing is returned; an error message is issued.

Author(s)

Thomas W. Yee

References

Yee, T. W. (2006). Constrained additive ordination. *Ecology*, **87**, 203–213.

Yee, T. W. and Hastie, T. J. (2003). Reduced-rank vector generalized linear models. *Statistical Modelling*, **3**, 15–41.

See Also

`rrvglm`, `cqo`.

Examples

```
## Not run:
clo()

## End(Not run)
```

cloglogink *Complementary Log-log Link Function*

Description

Computes the complementary log-log transformation, including its inverse and the first two derivatives. The complementary log transformation is also computed.

Usage

```
cloglogink(theta, bvalue = NULL, inverse = FALSE, deriv = 0,
            short = TRUE, tag = FALSE)
   cloglink(theta, bvalue = NULL, inverse = FALSE, deriv = 0,
            short = TRUE, tag = FALSE)
```

Arguments

`theta`	Numeric or character. See below for further details.
`bvalue`	See `Links` for general information about links.
`inverse, deriv, short, tag`	Details at `Links`.

Details

The complementary log-log link function is commonly used for parameters that lie in the unit interval. But unlike `logitlink`, `probitlink` and `cauchitlink`, this link is not symmetric. It is the inverse CDF of the extreme value (or Gumbel or log-Weibull) distribution. Numerical values of `theta` close to 0 or 1 or out of range result in `Inf`, `-Inf`, `NA` or `NaN`.

The complementary log link function is the same as the complementary log-log but the outer log is omitted. This link is suitable for `lrho` in `betabinomial` because it handles probability-like parameters but also allows slight negative values in theory. In particular, `cloglink` safeguards against parameters exceeding unity.

Value

For `deriv = 0`, the complimentary log-log of `theta`, i.e., `log(-log(1 - theta))` when `inverse = FALSE`, and if `inverse = TRUE` then `1-exp(-exp(theta))`.

For `deriv = 1`, then the function returns *d* `eta` / *d* `theta` as a function of `theta` if `inverse = FALSE`, else if `inverse = TRUE` then it returns the reciprocal.

Here, all logarithms are natural logarithms, i.e., to base e.

Note

Numerical instability may occur when `theta` is close to 1 or 0. One way of overcoming this is to use `bvalue`.

Changing 1s to 0s and 0s to 1s in the response means that effectively a loglog link is fitted. That is, tranform y by $1 - y$. That's why only one of `clogloglink` and `logloglink` is written.

With constrained ordination (e.g., `cqo` and `cao`) used with `binomialff`, a complementary log-log link function is preferred over the default `logitlink`, for a good reason. See the example below.

In terms of the threshold approach with cumulative probabilities for an ordinal response this link function corresponds to the extreme value distribution.

Author(s)

Thomas W. Yee

References

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*, 2nd ed. London: Chapman & Hall.

See Also

`Links`, `logitoffsetlink`, `logitlink`, `probitlink`, `cauchitlink`, `pgumbel`.

Examples

```
p <- seq(0.01, 0.99, by = 0.01)
clogloglink(p)
max(abs(clogloglink(clogloglink(p), inverse = TRUE) - p))  # Should be 0

p <- c(seq(-0.02, 0.02, by = 0.01), seq(0.97, 1.02, by = 0.01))
clogloglink(p)  # Has NAs
clogloglink(p, bvalue = .Machine$double.eps)  # Has no NAs

## Not run:
p <- seq(0.01, 0.99, by = 0.01)
plot(p, logitlink(p), type = "l", col = "limegreen", lwd = 2, las = 1,
     main = "Some probability link functions", ylab = "transformation")
lines(p, probitlink(p), col = "purple", lwd = 2)
lines(p, clogloglink(p), col = "chocolate", lwd = 2)
lines(p, cauchitlink(p), col = "tan", lwd = 2)
abline(v = 0.5, h = 0, lty = "dashed")
legend(0.1, 4, c("logitlink", "probitlink", "clogloglink", "cauchitlink"),
       col = c("limegreen", "purple", "chocolate", "tan"), lwd = 2)

## End(Not run)

## Not run:
# This example shows that clogloglink is preferred over logitlink
n <- 500; p <- 5; S <- 3; Rank <- 1  # Species packing model:
mydata <- rcqo(n, p, S, eq.tol = TRUE, es.opt = TRUE, eq.max = TRUE,
```

```
                family = "binomial", hi.abundance = 5, seed = 123,
                Rank = Rank)
fitc <- cqo(attr(mydata, "formula"), I.tol = TRUE, data = mydata,
            fam = binomialff(multiple.responses = TRUE, link = "cloglog"),
            Rank = Rank)
fitl <- cqo(attr(mydata, "formula"), I.tol = TRUE, data = mydata,
            fam = binomialff(multiple.responses = TRUE, link = "logitlink"),
            Rank = Rank)

# Compare the fitted models (cols 1 and 3) with the truth (col 2)
cbind(concoef(fitc), attr(mydata, "concoefficients"), concoef(fitl))

## End(Not run)
```

CM.equid *Constraint Matrices for Symmetry, Order, Parallelism, etc.*

Description

Given *M* linear/additive predictors, construct the constraint matrices to allow symmetry, (linear and normal) ordering, etc. in terms such as the intercept.

Usage

```
CM.equid(M, Trev = FALSE, Tref = 1)
 CM.free(M, Trev = FALSE, Tref = 1)
 CM.ones(M, Trev = FALSE, Tref = 1)
CM.symm0(M, Trev = FALSE, Tref = 1)
CM.symm1(M, Trev = FALSE, Tref = 1)
CM.qnorm(M, Trev = FALSE, Tref = 1)
```

Arguments

`M`	Number of linear/additive predictors, usually > 1.
`Tref`	Reference level for the threshold, this should be a single value from `1:M`. This argument is ignored by some of the above functions.
`Trev`	Logical. Apply reverse direction for the thresholds direction? This argument is ignored by some of the above functions.

Details

A constraint matrix is $M \times R$ where R is its rank and usually the elements are 0, 1 or -1. There is a constraint matrix for each column of the LM matrix used to fit the `vglm`. They are used to apportion the regression coefficients to the linear predictors, e.g., parallelism, exchangeability, etc. The functions described here are intended to construct constraint matrices easily for symmetry constraints and linear ordering etc. They are potentially useful for categorical data analysis (e.g., `cumulative`, `multinomial`), especially for the intercept term. When applied to `cumulative`, they are sometimes called *structured thresholds*, e.g., **ordinal**.

One example is the stereotype model proposed by Anderson (1984) (see `multinomial` and `rrvglm`) where the elements of the **A** matrix are ordered. This is not fully possible in **VGAM** but some special cases can be fitted, e.g., use `CM.equid` to create a linear ordering. And `CM.symm1` might result in fully ordered estimates too, etc.

`CM.free` creates *free* or unconstrained estimates. It is almost always the case for VGLMs, and is simply `diag(M)`.

`CM.ones` creates *equal* estimates, which is also known as the *parallelism* assumption in models such as `cumulative`. It gets its name because the constraint matrix is simply `matrix(1, M, 1)`.

`CM.equid` creates *equid*istant estimates. This is a linear scaling, and the direction and origin are controlled by `Treverse` and `Tref` respectively.

`CM.qnorm` and `CM.qlogis` are based on `qnorm` and `qlogis`. For example, `CM.qnorm(M)` is essentially `cbind(qnorm(seq(M) / (M + 1)))`. This might be useful with a model with `probitlink` applied to multiple intercepts.

Further details can be found at `cumulative` and `CommonVGAMffArguments`,

Value

A constraint matrix.

See Also

`CommonVGAMffArguments`, `cumulative`, `acat`, `cratio`, `sratio`, `multinomial`.

Examples

```
CM.equid(4)
CM.equid(4, Trev = TRUE, Tref = 3)
CM.symm1(5)
CM.symm0(5)
CM.qnorm(5)
```

`coalminers` *Breathlessness and Wheeze Amongst Coalminers Data*

Description

Coalminers who are smokers without radiological pneumoconiosis, classified by age, breathlessness and wheeze.

Usage

```
data(coalminers)
```

Format

A data frame with 9 age groups with the following 5 columns.

BW Counts with breathlessness and wheeze.

BnW Counts with breathlessness but no wheeze.

nBW Counts with no breathlessness but wheeze.

nBnW Counts with neither breathlessness or wheeze.

age Age of the coal miners (actually, the midpoints of the 5-year category ranges).

Details

The data were published in Ashford and Sowden (1970). A more recent analysis is McCullagh and Nelder (1989, Section 6.6).

Source

Ashford, J. R. and Sowden, R. R. (1970) Multi-variate probit analysis. *Biometrics*, **26**, 535–546.

References

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*. 2nd ed. London: Chapman & Hall.

Examples

```
str(coalminers)
```

`Coef` *Computes Model Coefficients and Quantities*

Description

`Coef` is a generic function which computes model coefficients from objects returned by modelling functions. It is an auxiliary function to `coef` that enables extra capabilities for some specific models.

Usage

```
Coef(object, ...)
```

Arguments

`object`	An object for which the computation of other types of model coefficients or quantities is meaningful.
`...`	Other arguments fed into the specific methods function of the model.

Details

This function can often be useful for `vglm` objects with just an intercept term in the RHS of the formula, e.g., `y ~ 1`. Then often this function will apply the inverse link functions to the parameters. See the example below.

For reduced-rank VGLMs, this function can return the **A**, **C** matrices, etc.

For quadratic and additive ordination models, this function can return ecological meaningful quantities such as tolerances, optimums, maximums.

Value

The value returned depends specifically on the methods function invoked.

Warning

This function may not work for *all* **VGAM** family functions. You should check your results on some artificial data before applying it to models fitted to real data.

Author(s)

Thomas W. Yee

References

Yee, T. W. and Hastie, T. J. (2003). Reduced-rank vector generalized linear models. *Statistical Modelling*, **3**, 15–41.

See Also

`coef`, `Coef.vlm`, `Coef.rrvglm`, `Coef.qrrvglm`, `depvar`.

Examples

```
nn <- 1000
bdata <- data.frame(y = rbeta(nn, shape1 = 1, shape2 = 3))  # Original scale
fit <- vglm(y ~ 1, betaR, data = bdata, trace = TRUE)  # Intercept-only model
coef(fit, matrix = TRUE)  # Both on a log scale
Coef(fit)  # On the original scale
```

`Coef.qrrvglm` *Returns Important Matrices etc. of a QO Object*

Description

This methods function returns important matrices etc. of a QO object.

Usage

```
Coef.qrrvglm(object, varI.latvar = FALSE, refResponse = NULL, ...)
```

Arguments

`object`	A CQO object. The former has class `"qrrvglm"`.
`varI.latvar`	Logical indicating whether to scale the site scores (latent variables) to have variance-covariance matrix equal to the rank-R identity matrix. All models have uncorrelated site scores (latent variables), and this option stretches or shrinks the ordination axes if `TRUE`. See below for further details.
`refResponse`	Integer or character. Specifies the *reference response* or *reference species*. By default, the reference species is found by searching sequentially starting from the first species until a positive-definite tolerance matrix is found. Then this tolerance matrix is transformed to the identity matrix. Then the sites scores (latent variables) are made uncorrelated. See below for further details.
`...`	Currently unused.

Details

If `I.tolerances=TRUE` or `eq.tolerances=TRUE` (and its estimated tolerance matrix is positive-definite) then all species' tolerances are unity by transformation or by definition, and the spread of the site scores can be compared to them. Vice versa, if one wishes to compare the tolerances with the sites score variability then setting `varI.latvar=TRUE` is more appropriate.

For rank-2 QRR-VGLMs, one of the species can be chosen so that the angle of its major axis and minor axis is zero, i.e., parallel to the ordination axes. This means the effect on the latent vars is independent on that species, and that its tolerance matrix is diagonal. The argument `refResponse` allows one to choose which is the reference species, which must have a positive-definite tolerance matrix, i.e., is bell-shaped. If `refResponse` is not specified, then the code will try to choose some reference species starting from the first species. Although the `refResponse` argument could possibly be offered as an option when fitting the model, it is currently available after fitting the model, e.g., in the functions `Coef.qrrvglm` and `lvplot.qrrvglm`.

Value

The **A**, **B1**, **C**, **T**, **D** matrices/arrays are returned, along with other slots. The returned object has class `"Coef.qrrvglm"` (see `Coef.qrrvglm-class`).

Note

Consider an equal-tolerances Poisson/binomial CQO model with `noRRR = ~ 1`. For $R = 1$ it has about $2S + p_2$ parameters. For $R = 2$ it has about $3S + 2p_2$ parameters. Here, S is the number of species, and $p_2 = p - 1$ is the number of environmental variables making up the latent variable. For an unequal-tolerances Poisson/binomial CQO model with `noRRR = ~ 1`, it has about $3S - 1 + p_2$ parameters for $R = 1$, and about $6S - 3 + 2p_2$ parameters for $R = 2$. Since the total number of data points is nS, where n is the number of sites, it pays to divide the number of data points by the number of parameters to get some idea about how much information the parameters contain.

Author(s)

Thomas W. Yee

References

Yee, T. W. (2004). A new technique for maximum-likelihood canonical Gaussian ordination. *Ecological Monographs*, **74**, 685–701.

Yee, T. W. (2006). Constrained additive ordination. *Ecology*, **87**, 203–213.

See Also

`cqo`, `Coef.qrrvglm-class`, `print.Coef.qrrvglm`, `lvplot.qrrvglm`.

Examples

```
set.seed(123)
x2 <- rnorm(n <- 100)
x3 <- rnorm(n)
x4 <- rnorm(n)
latvar1 <- 0 + x3 - 2*x4
lambda1 <- exp(3 - 0.5 * ( latvar1-0)^2)
lambda2 <- exp(2 - 0.5 * ( latvar1-1)^2)
lambda3 <- exp(2 - 0.5 * ((latvar1+4)/2)^2)  # Unequal tolerances
y1 <- rpois(n, lambda1)
y2 <- rpois(n, lambda2)
y3 <- rpois(n, lambda3)
set.seed(111)
# vvv p1 <- cqo(cbind(y1, y2, y3) ~ x2 + x3 + x4, poissonff, trace = FALSE)
## Not run:  lvplot(p1, y = TRUE, lcol = 1:3, pch = 1:3, pcol = 1:3)

# vvv Coef(p1)
# vvv print(Coef(p1), digits=3)
```

`Coef.qrrvglm-class` *Class "Coef.qrrvglm"*

Description

The most pertinent matrices and other quantities pertaining to a QRR-VGLM (CQO model).

Objects from the Class

Objects can be created by calls of the form `Coef(object,...)` where `object` is an object of class `"qrrvglm"` (created by `cqo`).

In this document, R is the *rank*, M is the number of linear predictors and n is the number of observations.

Slots

A: Of class "matrix", **A**, which are the linear 'coefficients' of the matrix of latent variables. It is M by R.

B1: Of class "matrix", **B1**. These correspond to terms of the argument noRRR.

C: Of class "matrix", **C**, the canonical coefficients. It has R columns.

Constrained: Logical. Whether the model is a constrained ordination model.

D: Of class "array", D[,,j] is an order-Rank matrix, for j = 1,...,M. Ideally, these are negative-definite in order to make the response curves/surfaces bell-shaped.

Rank: The rank (dimension, number of latent variables) of the RR-VGLM. Called R.

latvar: n by R matrix of latent variable values.

latvar.order: Of class "matrix", the permutation returned when the function order is applied to each column of latvar. This enables each column of latvar to be easily sorted.

Maximum: Of class "numeric", the M maximum fitted values. That is, the fitted values at the optimums for noRRR = ~ 1 models. If noRRR is not ~ 1 then these will be NAs.

NOS: Number of species.

Optimum: Of class "matrix", the values of the latent variables where the optimums are. If the curves are not bell-shaped, then the value will be NA or NaN.

Optimum.order: Of class "matrix", the permutation returned when the function order is applied to each column of Optimum. This enables each row of Optimum to be easily sorted.

bellshaped: Vector of logicals: is each response curve/surface bell-shaped?

dispersion: Dispersion parameter(s).

Dzero: Vector of logicals, is each of the response curves linear in the latent variable(s)? It will be if and only if D[,,j] equals **O**, for j = 1,...,M .

Tolerance: Object of class "array", Tolerance[,,j] is an order-Rank matrix, for j = 1,...,M, being the matrix of tolerances (squared if on the diagonal). These are denoted by **T** in Yee (2004). Ideally, these are positive-definite in order to make the response curves/surfaces bell-shaped. The tolerance matrices satisfy $T_s = -\frac{1}{2} D_s^{-1}$.

Author(s)

Thomas W. Yee

References

Yee, T. W. (2004). A new technique for maximum-likelihood canonical Gaussian ordination. *Ecological Monographs*, **74**, 685–701.

See Also

Coef.qrrvglm, cqo, print.Coef.qrrvglm.

Examples

```
x2 <- rnorm(n <- 100)
x3 <- rnorm(n)
x4 <- rnorm(n)
latvar1 <- 0 + x3 - 2*x4
lambda1 <- exp(3 - 0.5 * ( latvar1-0)^2)
lambda2 <- exp(2 - 0.5 * ( latvar1-1)^2)
lambda3 <- exp(2 - 0.5 * ((latvar1+4)/2)^2)
y1 <- rpois(n, lambda1)
y2 <- rpois(n, lambda2)
y3 <- rpois(n, lambda3)
yy <- cbind(y1, y2, y3)
# vvv p1 <- cqo(yy ~ x2 + x3 + x4, fam = poissonff, trace = FALSE)
## Not run:
lvplot(p1, y = TRUE, lcol = 1:3, pch = 1:3, pcol = 1:3)

## End(Not run)
# vvv print(Coef(p1), digits = 3)
```

Coef.rrvglm — *Returns Important Matrices etc. of a RR-VGLM Object*

Description

This methods function returns important matrices etc. of a RR-VGLM object.

Usage

```
Coef.rrvglm(object, ...)
```

Arguments

`object`	An object of class `"rrvglm"`.
`...`	Currently unused.

Details

The **A**, **B1**, **C** matrices are returned, along with other slots. See `rrvglm` for details about RR-VGLMs.

Value

An object of class `"Coef.rrvglm"` (see `Coef.rrvglm-class`).

Note

This function is an alternative to `coef.rrvglm`.

Author(s)

Thomas W. Yee

References

Yee, T. W. and Hastie, T. J. (2003). Reduced-rank vector generalized linear models. *Statistical Modelling*, **3**, 15–41.

See Also

`Coef.rrvglm-class`, `print.Coef.rrvglm`, `rrvglm`.

Examples

```
# Rank-1 stereotype model of Anderson (1984)
pneumo <- transform(pneumo, let = log(exposure.time), x3 = runif(nrow(pneumo)))
fit <- rrvglm(cbind(normal, mild, severe) ~ let + x3, multinomial, data = pneumo)
coef(fit, matrix = TRUE)
Coef(fit)
```

`Coef.rrvglm-class` *Class "Coef.rrvglm"*

Description

The most pertinent matrices and other quantities pertaining to a RR-VGLM.

Objects from the Class

Objects can be created by calls of the form `Coef(object, ...)` where `object` is an object of class `rrvglm` (see `rrvglm-class`).

In this document, M is the number of linear predictors and n is the number of observations.

Slots

`A`**:** Of class `"matrix"`, **A**.

`B1`**:** Of class `"matrix"`, **B1**.

`C`**:** Of class `"matrix"`, **C**.

`Rank`**:** The rank of the RR-VGLM.

`colx1.index`**:** Index of the columns of the `"vlm"`-type model matrix corresponding to the variables in **x1**. These correspond to **B1**.

`colx2.index`**:** Index of the columns of the `"vlm"`-type model matrix corresponding to the variables in **x2**. These correspond to the reduced-rank regression.

`Atilde`**:** Object of class `"matrix"`, the **A** matrix with the corner rows removed. Thus each of the elements have been estimated. This matrix is returned only if corner constraints were used.

Author(s)

Thomas W. Yee

References

Yee, T. W. and Hastie, T. J. (2003). Reduced-rank vector generalized linear models. *Statistical Modelling*, **3**, 15–41.

See Also

`Coef.rrvglm`, `rrvglm`, `rrvglm-class`, `print.Coef.rrvglm`.

Examples

```
# Rank-1 stereotype model of Anderson (1984)
pneumo <- transform(pneumo, let = log(exposure.time), x3 = runif(nrow(pneumo)))
fit <- rrvglm(cbind(normal, mild, severe) ~ let + x3, multinomial, data = pneumo)
coef(fit, matrix = TRUE)
Coef(fit)
# print(Coef(fit), digits = 3)
```

`Coef.vlm` *Extract Model Coefficients for VLM Objects*

Description

Amongst other things, this function applies inverse link functions to the parameters of intercept-only VGLMs.

Usage

```
Coef.vlm(object, ...)
```

Arguments

`object`	A fitted model.
`...`	Arguments which may be passed into `coef`.

Details

Most **VGAM** family functions apply a link function to the parameters, e.g., positive parameter are often have a log link, parameters between 0 and 1 have a logit link. This function can back-transform the parameter estimate to the original scale.

Value

For intercept-only models (e.g., formula is `y ~ 1`) the back-transformed parameter estimates can be returned.

Warning

This function may not work for *all* **VGAM** family functions. You should check your results on some artificial data before applying it to models fitted to real data.

Author(s)

Thomas W. Yee

References

Yee, T. W. and Hastie, T. J. (2003). Reduced-rank vector generalized linear models. *Statistical Modelling*, **3**, 15–41.

See Also

`Coef`, `coef`.

Examples

```
set.seed(123); nn <- 1000
bdata <- data.frame(y = rbeta(nn, shape1 = 1, shape2 = 3))
fit <- vglm(y ~ 1, betaff, data = bdata, trace = TRUE)  # intercept-only model
coef(fit, matrix = TRUE)  # log scale
Coef(fit)  # On the original scale
```

`coefvgam` *Extract Model Coefficients of a vgam() Object*

Description

Extracts the estimated coefficients from vgam() objects.

Usage

```
coefvgam(object, type = c("linear", "nonlinear"), ...)
```

Arguments

`object`	A `vgam` object.
`type`	Character. The default is the first choice.
`...`	Optional arguments fed into `coefvlm`.

Details

For VGAMs, because modified backfitting is performed, each fitted function is decomposed into a linear and nonlinear (smooth) part. The argument `type` is used to return which one is wanted.

Value

A vector if type = "linear". A list if type = "nonlinear", and each component of this list corresponds to an s term; the component contains an S4 object with slot names such as "Bcoefficients", "knots", "xmin", "xmax".

Author(s)

Thomas W. Yee

See Also

vgam, coefvlm, coef.

Examples

```
fit <- vgam(agaaus ~ s(altitude, df = 2), binomialff, data = hunua)
coef(fit)  # Same as coef(fit, type = "linear")
(ii <- coef(fit, type = "nonlinear"))
is.list(ii)
names(ii)
slotNames(ii[[1]])
```

---

coefvlm — *Extract Model Coefficients*

Description

Extracts the estimated coefficients from VLM objects such as VGLMs.

Usage

```
coefvlm(object, matrix.out = FALSE, label = TRUE, colon = FALSE, ...)
```

Arguments

object	An object for which the extraction of coefficients is meaningful. This will usually be a vglm object.
matrix.out	Logical. If TRUE then a matrix is returned. The explanatory variables are the rows. The linear/additive predictors are the columns. The constraint matrices are used to compute this matrix.
label	Logical. If FALSE then the names of the vector of coefficients are set to NULL.
colon	Logical. Explanatory variables which appear in more than one linear/additive predictor are labelled with a colon, e.g., age:1, age:2. However, if it only appears in one linear/additive predictor then the :1 is omitted by default. Then setting colon = TRUE will add the :1.
...	Currently unused.

Details

This function works in a similar way to applying `coef()` to a `lm` or `glm` object. However, for VGLMs, there are more options available.

Value

A vector usually. A matrix if `matrix.out = TRUE`.

Author(s)

Thomas W. Yee

References

Yee, T. W. and Hastie, T. J. (2003). Reduced-rank vector generalized linear models. *Statistical Modelling*, **3**, 15–41.

See Also

`vglm`, `coefvgam`, `coef`.

Examples

```
zdata <- data.frame(x2 = runif(nn <- 200))
zdata <- transform(zdata, pstr0  = logitlink(-0.5 + 1*x2, inverse = TRUE),
                          lambda =   loglink( 0.5 + 2*x2, inverse = TRUE))
zdata <- transform(zdata, y2 = rzipois(nn, lambda, pstr0 = pstr0))

fit2 <- vglm(y2 ~ x2, zipoisson(zero = 1), data = zdata, trace = TRUE)
coef(fit2, matrix = TRUE)  # Always a good idea
coef(fit2)
coef(fit2, colon = TRUE)
```

`CommonVGAMffArguments` *Common VGAM Family Function Arguments*

Description

Here is a description of some common and typical arguments found in many **VGAM** family functions, e.g., `zero`, `lsigma`, `isigma`, `gsigma`, `eq.mean`, `nsimEI` and `parallel`.

Usage

```
TypicalVGAMfamilyFunction(lsigma = "loglink",
                          isigma = NULL,
                          zero = NULL, gsigma = exp(-5:5),
                          eq.mean = FALSE,
                          parallel = TRUE,
```

```
imethod = 1,
vfl = FALSE, Form2 = NULL,
type.fitted = c("mean", "quantiles", "Qlink",
                "pobs0", "pstr0", "onempstr0"),
percentiles = c(25, 50, 75),
probs.x = c(0.15, 0.85),
probs.y = c(0.25, 0.50, 0.75),
multiple.responses = FALSE, earg.link = FALSE,
ishrinkage = 0.95, nointercept = NULL,
whitespace = FALSE, bred = FALSE, lss = TRUE,
oim = FALSE, nsimEIM = 100, byrow.arg = FALSE,
link.list = list("(Default)" = "identitylink",
                 x2          = "loglink",
                 x3          = "logofflink",
                 x4          = "multilogitlink",
                 x5          = "multilogitlink"),
earg.list = list("(Default)" = list(),
                 x2          = list(),
                 x3          = list(offset = -1),
                 x4          = list(),
                 x5          = list()),
Thresh = NULL, nrfs = 1)
```

Arguments

`lsigma` Character. Link function applied to a parameter and not necessarily a mean. See `Links` for a selection of choices. If there is only one parameter then this argument is often called `link`.

`isigma` Optional initial values can often be inputted using an argument beginning with `"i"`. For example, `"isigma"` and `"ilocation"`, or just `"init"` if there is one parameter. A value of `NULL` means a value is computed internally, i.e., a *self-starting* **VGAM** family function. If a failure to converge occurs make use of these types of arguments.

`zero` An important argument, either an integer vector, or a vector of character strings. If an integer, then it specifies which linear/additive predictor is modelled as *intercept-only*. That is, the regression coefficients are set to zero for all covariates except for the intercept. If `zero` is specified then it may be a vector with values from the set $\{1, 2, \ldots, M\}$. The value `zero = NULL` means model *all* linear/additive predictors as functions of the explanatory variables. Here, M is the number of linear/additive predictors. Technically, if `zero` contains the value j then the jth row of every constraint matrix (except for the intercept) consists of all 0 values.

Some **VGAM** family functions allow the `zero` argument to accept negative values; if so then its absolute value is recycled over each (usual) response. For example, `zero = -2` for the two-parameter negative binomial distribution would mean, for each response, the second linear/additive predictor is modelled as intercepts-only. That is, for all the k parameters in `negbinomial` (this **VGAM** family function can handle a matrix of responses).

Suppose zero = zerovec where zerovec is a vector of negative values. If G is the usual M value for a univariate response then the actual values for argument zero are all values in c(abs(zerovec), G + abs(zerovec), 2*G + abs(zerovec), ...) lying in the integer range 1 to M. For example, setting zero = -c(2, 3) for a matrix response of 4 columns with zinegbinomial (which usually has $G = M = 3$ for a univariate response) would be equivalent to zero = c(2, 3, 5, 6, 8, 9, 11, 12). This example has $M = 12$. Note that if zerovec contains negative values then their absolute values should be elements from the set 1:G.

Note: zero may have positive and negative values, for example, setting zero = c(-2, 3) in the above example would be equivalent to zero = c(2, 3, 5, 8, 11).

The argument zero also accepts a character vector (for **VGAM** 1.0-1 onwards). Each value is fed into grep with fixed = TRUE, meaning that wildcards "*" are not useful. See the example below—all the variants work; those with LOCAT issue a warning that that value is unmatched. Importantly, the parameter names are c("location1", "scale1", "location2","scale2") because there are 2 responses. Yee (2015) described zero for only numerical input. Allowing character input is particularly important when the number of parameters cannot be determined without having the actual data first. For example, with time series data, an ARMA(p,q) process might have parameters $\theta_1, \ldots, \theta_p$ which should be intercept-only by default. Then specifying a numerical default value for zero would be too difficult (there are the drift and scale parameters too). However, it is possible with the character representation: zero = "theta" would achieve this. In the future, most **VGAM** family functions might be converted to the character representation—the advantage being that it is more readable. When programming a **VGAM** family function that allows character input, the variable predictors.names must be assigned correctly.

If the constraints argument is used then the family function's zero argument (if it exists) needs to be set to NULL. This avoids what could be a probable contradiction. Sometimes setting other arguments related to constraint matrices to FALSE is also a good idea, e.g., parallel = FALSE, exchangeable = FALSE.

gsigma
: Grid-search initial values can be inputted using an argument beginning with "g", e.g., "gsigma", "gshape" and "gscale". If argument isigma is inputted then that has precedence over gsigma, etc. If the grid search is 2-dimensional then it is advisable not to make the vectors too long as a nested for loop may be used. Ditto for 3-dimensions etc. Sometimes a ".mux" is added as a suffix, e.g., gshape.mux; this means that the grid is created relatively and not absolutely, e.g., its values are multipled by some single initial estimate of the parameter in order to create the grid on an absolute scale.

  Some family functions have an argument called gprobs.y. This is fed into the probs argument of quantile in order to obtain some values of central tendency of the response, i.e., some spread of values in the middle. when imethod = 1 to obtain an initial value for the mean Some family functions have an argument called iprobs.y, and if so, then these values can overwrite gprobs.y.

eq.mean
: Logical. Constrain all the means to be equal? This type of argument is simpler than parallel because only a single TRUE or FALSE can be assigned and not a formula. Thus if TRUE then it will be enforced over all variables.

`parallel` A logical, or a simple formula specifying which terms have equal/unequal coefficients. The formula must be simple, i.e., additive with simple main effects terms. Interactions and nesting etc. are not handled. To handle complex formulas use the `constraints` argument (of `vglm` etc.); however, there is a lot more setting up involved and things will not be as convenient.

Here are some examples. 1. `parallel = TRUE ~ x2 + x5` means the parallelism assumption is only applied to X_2, X_5 and the intercept. 2. `parallel = TRUE ~ -1` and `parallel = TRUE ~ 0` mean the parallelism assumption is applied to *no* variables at all. Similarly, `parallel = FALSE ~ -1` and `parallel = FALSE ~ 0` mean the parallelism assumption is applied to *all* the variables including the intercept. 3. `parallel = FALSE ~ x2 - 1` and `parallel = FALSE ~ x2 + 0` applies the parallelism constraint to all terms (including the intercept) except for X_2.

This argument is common in **VGAM** family functions for categorical responses, e.g., `cumulative`, `acat`, `cratio`, `sratio`. For the proportional odds model (`cumulative`) having parallel constraints applied to each explanatory variable (except for the intercepts) means the fitted probabilities do not become negative or greater than 1. However this parallelism or proportional-odds assumption ought to be checked.

`nsimEIM` Some **VGAM** family functions use simulation to obtain an approximate expected information matrix (EIM). For those that do, the `nsimEIM` argument specifies the number of random variates used per observation; the mean of `nsimEIM` random variates is taken. Thus `nsimEIM` controls the accuracy and a larger value may be necessary if the EIMs are not positive-definite. For intercept-only models (`y ~ 1`) the value of `nsimEIM` can be smaller (since the common value used is also then taken as the mean over the observations), especially if the number of observations is large.

Some **VGAM** family functions provide two algorithms for estimating the EIM. If applicable, set `nsimEIM = NULL` to choose the other algorithm.

`imethod` An integer with value `1` or `2` or `3` or ... which specifies the initialization method for some parameters or a specific parameter. If failure to converge occurs try the next higher value, and continue until success. For example, `imethod = 1` might be the method of moments, and `imethod = 2` might be another method. If no value of `imethod` works then it will be necessary to use arguments such as `isigma`. For many **VGAM** family functions it is advisable to try this argument with all possible values to safeguard against problems such as converging to a local solution. **VGAM** family functions with this argument usually correspond to a model or distribution that is relatively hard to fit successfully, therefore care is needed to ensure the global solution is obtained. So using all possible values that this argument supplies is a good idea.

VGAM family functions such `genpoisson2` recycle `imethod` to be of length 2 corresponding to the 2 parameters. In the future, this feature will be extended to other family functions to confer more flexibility.

`Form2` Formula. Using applied to models with $M = 2$. Specifies the terms for η_2 and the other terms belong to η_1. It is a way to partition the **X** matrix into two sets of covariates, where they are assigned to each η_j separately. This argument sets up constraint matrices `rbind(0, 1)` for terms in `Form2` and `rbind(1, 0)` for setdiff(formula, Form2) so to speak. Note that sometimes this argument is only accessed if vfl = TRUE. Arguments such as Form1 and Form3 are also possible in **VGAM** family functions because the η_j which is likely to be modelled more simply is chosen for convenience.

vfl
: A single logical. This stands for *variance–variance factored loglinear* (VFL) model. If TRUE then usually some other argument such as Form2 or parallel is used to partition the main vglm formula into two sets of covariates. For some families such as negbinomial this enables overdispersion to be modelled conveniently via a loglinear model, given the mean. It is necessary to read the online help regarding each **VGAM** family function because each one may different from others. To fit some VFL models it is necessary to make a copy of existing covariates by creating new variable names and then adding it to the main formula.

  A good question is: why is vfl necessary? Wouldn't Form2 be sufficient? Setting vfl = TRUE enables some internal checking such as avoiding conflicts. For example, it is often necessary to set zero = NULL and parallel = FALSE, otherwise there would be contradictions.

type.fitted
: Character. Type of fitted value returned by the fitted() methods function. The first choice is always the default. The available choices depends on what kind of family function it is. Using the first few letters of the chosen choice is okay. See fittedvlm for more details.

  The choice "Qlink" refers to quantile-links, which was introduced in December 2018 in **VGAMextra** 0.0-2 for several 1-parameter distributions. Here, either the loglink or logitlink or identitylink of the quantile is the link function (and the choice is dependent on the support of the distribution), and link functions end in "Qlink". A limited amount of support is provided for such links, e.g., fitted(fit) are the fitted quantiles, which is the same as predict(fit, type = "response"). However, fitted(fit, percentiles = 77) will not work.

percentiles
: Numeric vector, with values between 0 and 100 (although it is not recommended that exactly 0 or 100 be inputted). Used only if type.fitted = "quantiles" or type.fitted = "percentiles", then this argument specifies the values of these quantiles. The argument name tries to reinforce that the values lie between 0 and 100. See fittedvlm for more details.

probs.x, probs.y
: Numeric, with values in (0, 1). The probabilites that define quantiles with respect to some vector, usually an x or y of some sort. This is used to create two subsets of data corresponding to 'low' and 'high' values of x or y. Each value is separately fed into the probs argument of quantile. If the data set size is small then it may be necessary to increase/decrease slightly the first/second values respectively.

lss
: Logical. This stands for the ordering: location, scale and shape. Should the ordering of the parameters be in this order? Almost all **VGAM** family functions have this order by default, but in order to match the arguments of existing R functions, one might need to set lss = FALSE. For example, the arguments of weibullR are scale and shape, whereas rweibull are shape and scale. As a temporary measure (from **VGAM** 0.9-7 onwards but prior to version 1.0-0),

some family functions such as `sinmad` have an `lss` argument without a default. For these, setting `lss = FALSE` will work. Later, `lss = TRUE` will be the default. Be careful for the `dpqr`-type functions, e.g., `rsinmad`.

`Thresh` *Thresholds* is another name for the intercepts, e.g., for categorical models. They may be constrained by functions such as `CM.equid` and `CM.qnorm`. The string `"CM."` and the `Thresh` argument is pasted and then that function is called to obtain the constraint matrix for the intercept term. So `Thresh = "free"`, `Thresh = "equid"`, `Thresh = "qnorm"`, `Thresh = "symm0"`, `Thresh = "symm1"` etc. are possibilities. Families that use this include `multinomial`, `cratio`, `sratio`, `cumulative`, `acat`.

`whitespace` Logical. Should white spaces (`" "`) be used in the labelling of the linear/additive predictors? Setting `TRUE` usually results in more readability but it occupies more columns of the output.

`oim` Logical. Should the observed information matrices (OIMs) be used for the working weights? In general, setting `oim = TRUE` means the Newton-Raphson algorithm, and `oim = FALSE` means Fisher-scoring. The latter uses the EIM, and is usually recommended. If `oim = TRUE` then `nsimEIM` is ignored.

`nrfs` Numeric, a value in $[0, 1]$. Experimental argument for allowing a mixture of Newton-Raphson and Fisher scoring. The working weights are taken as a linear combination of the two. If `nrfs = 0` then Newton-Raphson is used, i.e., the OIM is wholly used. If `nrfs = 1` then Fisher scoring is used, i.e., the EIM is wholly used. If convergence is successful then one might expect Newton-Raphson to be faster than Fisher scoring because the former has an order-2 convergence rate while the latter has a linear rate.

,

`multiple.responses`
Logical. Some **VGAM** family functions allow a multivariate or vector response. If so, then usually the response is a matrix with columns corresponding to the individual response variables. They are all fitted simultaneously. Arguments such as `parallel` may then be useful to allow for relationships between the regressions of each response variable. If `multiple.responses = TRUE` then sometimes the response is interpreted differently, e.g., `posbinomial` chooses the first column of a matrix response as success and combines the other columns as failure, but when `multiple.responses = TRUE` then each column of the response matrix is the number of successes and the `weights` argument is of the same dimension as the response and contains the number of trials.

`earg.link` This argument should be generally ignored.

`byrow.arg` Logical. Some **VGAM** family functions that handle multiple responses have arguments that allow input to be fed in which affect all the responses, e.g., `imu` for initalizing a `mu` parameter. In such cases it is sometime more convenient to input one value per response by setting `byrow.arg = TRUE`; then values are recycled in order to form a matrix of the appropriate dimension. This argument matches `byrow` in `matrix`; in fact it is fed into such using `matrix(..., byrow = byrow.arg)`. This argument has no effect when there is one response.

`ishrinkage` Shrinkage factor s used for obtaining initial values. Numeric, between 0 and 1. In general, the formula used is something like $s\mu + (1-s)y$ where μ is a measure of central tendency such as a weighted mean or median, and y is the response vector. For example, the initial values are slight perturbations of the mean towards the actual data. For many types of models this method seems to work well and is often reasonably robust to outliers in the response. Often this argument is only used if the argument `imethod` is assigned a certain value.

`nointercept` An integer-valued vector specifying which linear/additive predictors have no intercepts. Any values must be from the set $\{1,2,\ldots,M\}$. A value of `NULL` means no such constraints.

`bred` Logical. Some **VGAM** family functions will allow bias-reduction based on the work by Kosmidis and Firth. Sometimes half-stepping is a good idea; set `stepsize = 0.5` and monitor convergence by setting `trace = TRUE`.

`link.list, earg.list`
Some **VGAM** family functions (such as `normal.vcm`) implement models with potentially lots of parameter link functions. These two arguments allow many such links and extra arguments to be inputted more easily. One has something like `link.list = list("(Default)" = "identitylink", x2 = "loglink", x3 = "logofflink")` and `earg.list = list("(Default)" = list(), x2 = list(), x3 = "list(offset = -1)")`. Then any unnamed terms will have the default link with its corresponding extra argument. Note: the `multilogitlink` link is also possible, and if so, at least two instances of it are necessary. Then the last term is the baseline/reference group.

Details

Full details will be given in documentation yet to be written, at a later date! A general recommendation is to set `trace = TRUE` whenever any model fitting is done, since monitoring convergence is usually very informative.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Warning

The `zero` argument is supplied for convenience but conflicts can arise with other arguments, e.g., the `constraints` argument of `vglm` and `vgam`. See Example 5 below for an example. If not sure, use, e.g., `constraints(fit)` and `coef(fit, matrix = TRUE)` to check the result of a fit `fit`.

The arguments `zero` and `nointercept` can be inputted with values that fail. For example, `multinomial(zero = 2, nointercept = 1:3)` means the second linear/additive predictor is identically zero, which will cause a failure.

Be careful about the use of other potentially contradictory constraints, e.g., `multinomial(zero = 2, parallel = TRUE ~ x3)`. If in doubt, apply `constraints()` to the fitted object to check.

VGAM family functions with the `nsimEIM` may have inaccurate working weight matrices. If so, then the standard errors of the regression coefficients may be inaccurate. Thus output from `summary(fit)`, `vcov(fit)`, etc. may be misleading.

Changes relating to the `lss` argument have very important consequences and users must beware. Good programming style is to rely on the argument names and not on the order.

Author(s)

T. W. Yee

References

Yee, T. W. (2015). Vector Generalized Linear and Additive Models: With an Implementation in R. New York, USA: *Springer*.

Kosmidis, I. and Firth, D. (2009). Bias reduction in exponential family nonlinear models. *Biometrika*, **96**, 793–804.

Miranda-Soberanis, V. F. and Yee, T. W. (2019). New link functions for distribution–specific quantile regression based on vector generalized linear and additive models. *Journal of Probability and Statistics*, **5**, 1–11.

See Also

Links, vglm, vgam, vglmff-class, UtilitiesVGAM, multilogitlink, multinomial, **VGAMextra**.

Examples

```
# Example 1
cumulative()
cumulative(link = "probitlink", reverse = TRUE, parallel = TRUE)

# Example 2
wdata <- data.frame(x2 = runif(nn <- 1000))
wdata <- transform(wdata,
         y = rweibull(nn, shape = 2 + exp(1 + x2), scale = exp(-0.5)))
fit <- vglm(y ~ x2, weibullR(lshape = logofflink(offset = -2), zero = 2),
            data = wdata)
coef(fit, mat = TRUE)

# Example 3; multivariate (multiple) response
## Not run:
ndata <- data.frame(x = runif(nn <- 500))
ndata <- transform(ndata,
           y1 = rnbinom(nn, exp(1), mu = exp(3+x)),  # k is size
           y2 = rnbinom(nn, exp(0), mu = exp(2-x)))
fit <- vglm(cbind(y1, y2) ~ x, negbinomial(zero = -2), ndata)
coef(fit, matrix = TRUE)

## End(Not run)
# Example 4
## Not run:
# fit1 and fit2 are equivalent
fit1 <- vglm(ymatrix ~ x2 + x3 + x4 + x5,
             cumulative(parallel = FALSE ~ 1 + x3 + x5), cdata)
fit2 <- vglm(ymatrix ~ x2 + x3 + x4 + x5,
             cumulative(parallel = TRUE ~ x2 + x4), cdata)
```

```
## End(Not run)

# Example 5
udata <- data.frame(x2 = rnorm(nn <- 200))
udata <- transform(udata,
           x1copy = 1,  # Copy of the intercept
           x3 = runif(nn),
           y1 = rnorm(nn, 1 - 3*x2, sd = exp(1 + 0.2*x2)),
           y2 = rnorm(nn, 1 - 3*x2, sd = exp(1)))
args(uninormal)
fit1 <- vglm(y1 ~ x2, uninormal, udata)            # This is okay
fit2 <- vglm(y2 ~ x2, uninormal(zero = 2), udata)  # This is okay
fit4 <- vglm(y2 ~ x2 + x1copy + x3,
             uninormal(zero = NULL, vfl = TRUE,
                       Form2 = ~ x1copy + x3 - 1), udata)
coef(fit4,  matrix = TRUE)  # VFL model

# This creates potential conflict
clist <- list("(Intercept)" = diag(2), "x2" = diag(2))
fit3 <- vglm(y2 ~ x2, uninormal(zero = 2), data = udata,
             constraints = clist)  # Conflict!
coef(fit3, matrix = TRUE)  # Shows that clist[["x2"]] was overwritten,
constraints(fit3)  # i.e., 'zero' seems to override the 'constraints' arg

# Example 6 ('whitespace' argument)
pneumo <- transform(pneumo, let = log(exposure.time))
fit1 <- vglm(cbind(normal, mild, severe) ~ let,
             sratio(whitespace = FALSE, parallel = TRUE), pneumo)
fit2 <- vglm(cbind(normal, mild, severe) ~ let,
             sratio(whitespace = TRUE,  parallel = TRUE), pneumo)
head(predict(fit1), 2)  # No white spaces
head(predict(fit2), 2)  # Uses white spaces

# Example 7 ('zero' argument with character input)
set.seed(123); n <- 1000
ldata <- data.frame(x2 = runif(n))
ldata <- transform(ldata, y1 = rlogis(n, loc = 5*x2, scale = exp(2)))
ldata <- transform(ldata, y2 = rlogis(n, loc = 5*x2, scale = exp(1*x2)))
ldata <- transform(ldata, w1 = runif(n))
ldata <- transform(ldata, w2 = runif(n))
fit7 <- vglm(cbind(y1, y2) ~ x2,
#        logistic(zero = "location1"),  # location1 is intercept-only
#        logistic(zero = "location2"),
#        logistic(zero = "location*"),  # Not okay... all is unmatched
#        logistic(zero = "scale1"),
#        logistic(zero = "scale2"),
#        logistic(zero = "scale"),  # Both scale parameters are matched
         logistic(zero = c("location", "scale2")),  # All but scale1
#        logistic(zero = c("LOCAT", "scale2")),  # Only scale2 is matched
#        logistic(zero = c("LOCAT")),  # Nothing is matched
#        trace = TRUE,
#        weights = cbind(w1, w2),
         weights = w1,
```

```
          data = ldata)
coef(fit7, matrix = TRUE)
```

---

concoef *Extract Model Constrained/Canonical Coefficients*

Description

concoef is a generic function which extracts the constrained (canonical) coefficients from objects returned by certain modelling functions.

Usage

```
concoef(object, ...)
```

Arguments

object	An object for which the extraction of canonical coefficients is meaningful.
...	Other arguments fed into the specific methods function of the model.

Details

For constrained quadratic and ordination models, *canonical coefficients* are the elements of the **C** matrix used to form the latent variables. They are highly interpretable in ecology, and are looked at as weights or loadings.

They are also applicable for reduced-rank VGLMs.

Value

The value returned depends specifically on the methods function invoked.

Warning

concoef replaces ccoef; the latter is deprecated.

For QO models, there is a direct inverse relationship between the scaling of the latent variables (site scores) and the tolerances. One normalization is for the latent variables to have unit variance. Another normalization is for all the species' tolerances to be unit (provided eq.tolerances is TRUE). These two normalizations cannot simultaneously hold in general. For rank R models with $R > 1$ it becomes more complicated because the latent variables are also uncorrelated. An important argument when fitting quadratic ordination models is whether eq.tolerances is TRUE or FALSE. See Yee (2004) for details.

Author(s)

Thomas W. Yee

References

Yee, T. W. and Hastie, T. J. (2003). Reduced-rank vector generalized linear models. *Statistical Modelling*, **3**, 15–41.

Yee, T. W. (2004). A new technique for maximum-likelihood canonical Gaussian ordination. *Ecological Monographs*, **74**, 685–701.

Yee, T. W. (2006). Constrained additive ordination. *Ecology*, **87**, 203–213.

See Also

concoef-method, concoef.qrrvglm, concoef.cao, coef.

Examples

```
## Not run:  set.seed(111)  # This leads to the global solution
hspider[,1:6] <- scale(hspider[,1:6])  # Standardized environmental vars
p1 <- cqo(cbind(Alopacce, Alopcune, Alopfabr, Arctlute, Arctperi,
                Auloalbi, Pardlugu, Pardmont, Pardnigr, Pardpull,
                Trocterr, Zoraspin) ~
          WaterCon + BareSand + FallTwig + CoveMoss + CoveHerb + ReflLux,
          family = poissonff, data = hspider, Crow1positive = FALSE)
concoef(p1)

## End(Not run)
```

---

concoef-methods *Constrained (Canonical) Coefficients*

Description

concoef is a generic function used to return the constrained (canonical) coefficients of a constrained ordination model. The function invokes particular methods which depend on the class of the first argument.

Methods

object The object from which the constrained coefficients are extracted.

confintvglm *Confidence Intervals for Parameters of VGLMs*

Description

Computes confidence intervals (CIs) for one or more parameters in a fitted model. Currently the object must be a "vglm" object.

Usage

```
confintvglm(object, parm, level = 0.95, method = c("wald", "profile"),
            trace = NULL, ...)
```

Arguments

`object`	A fitted model object.
`parm, level, ...`	Same as `confint`.
`method`	Character. The default is the first method. Abbreviations are allowed. Currently "profile" is basically working; and it is likely to be more accurate especially for small samples, as it is based on a profile log likelihood, however it is computationally intensive.
`trace`	Logical. If `TRUE` then one can monitor the computation as it progresses (because it is expensive). The default is the orginal model's `trace` value (see `vglm.control`). Setting `FALSE` suppresses all intermediate output.

Details

The default for this methods function is based on `confint.default` and assumes asymptotic normality. In particular, the `coef` and `vcov` methods functions are used for `vglm-class` objects.

When `method = "profile"` the function `profilevglm` is called to do the profiling. The code is very heavily based on `profile.glm` which was originally written by D. M. Bates and W. N. Venables (For S in 1996) and subsequently corrected by B. D. Ripley. Sometimes the profiling method can give problems, for example, `cumulative` requires the M linear predictors not to intersect in the data cloud. Such numerical problems are less common when `method = "wald"`, however, it is well-known that inference based on profile likelihoods is generally more accurate than Wald, especially when the sample size is small. The deviance (`deviance(object)`) is used if possible, else the difference `2 * (logLik(object) - ell)` is computed, where `ell` are the values of the loglikelihood on a grid.

For Wald CIs and `rrvglm-class` objects, currently an error message is produced because I haven't gotten around to write the methods function; it's not too hard, but am too busy! An interim measure is to coerce the object into a "vglm" object, but then the confidence intervals will tend to be too narrow because the estimated constraint matrices are treated as known.

For Wald CIs and `vgam-class` objects, currently an error message is produced because the theory is undeveloped.

Value

Same as confint.

Note

The order of the values of argument method may change in the future without notice. The functions plot.profile.glm and pairs.profile.glm from **MASS** appear to work with output from this function.

Author(s)

Thomas Yee adapted confint.lm to handle "vglm" objects, for Wald-type confidence intervals. Also, profile.glm was originally written by D. M. Bates and W. N. Venables (For S in 1996) and subsequently corrected by B. D. Ripley. This function effectively calls confint.profile.glm() in **MASS**.

See Also

vcovvlm, summaryvglm, confint, profile.glm, lrt.stat.vlm, wald.stat, plot.profile.glm, pairs.profile.glm.

Examples

```
# Example 1: this is based on a glm example
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3, 1, 9); treatment <- gl(3, 3)
 glm.D93 <-  glm(counts ~ outcome + treatment, family = poisson())
vglm.D93 <- vglm(counts ~ outcome + treatment, family = poissonff)
confint(glm.D93) # needs MASS to be present on the system
confint.default(glm.D93)  # based on asymptotic normality
confint(vglm.D93)
confint(vglm.D93) - confint(glm.D93)    # Should be all 0s
confint(vglm.D93) - confint.default(glm.D93)  # based on asympt. normality

# Example 2: simulated negative binomial data with multiple responses
ndata <- data.frame(x2 = runif(nn <- 100))
ndata <- transform(ndata, y1 = rnbinom(nn, mu = exp(3+x2), size = exp(1)),
                          y2 = rnbinom(nn, mu = exp(2-x2), size = exp(0)))
fit1 <- vglm(cbind(y1, y2) ~ x2, negbinomial, data = ndata, trace = TRUE)
coef(fit1)
coef(fit1, matrix = TRUE)
confint(fit1)
confint(fit1, "x2:1")  #  This might be improved to "x2" some day...
## Not run:
confint(fit1, method = "profile")  # Computationally expensive
confint(fit1, "x2:1", method = "profile", trace = FALSE)

## End(Not run)

fit2 <- rrvglm(y1 ~ x2, negbinomial(zero = NULL), data = ndata)
confint(as(fit2, "vglm"))  # Too narrow (SEs are biased downwards)
```

`constraints` *Constraint Matrices*

Description

Extractor function for the *constraint matrices* of objects in the **VGAM** package.

Usage

```
constraints(object, ...)
constraints.vlm(object, type = c("lm", "term"), all = TRUE, which,
                matrix.out = FALSE, colnames.arg = TRUE,
                rownames.arg = TRUE, ...)
```

Arguments

- `object` Some **VGAM** object, for example, having class `vglmff-class`.
- `type` Character. Whether LM- or term-type constraints are to be returned. The number of such matrices returned is equal to `nvar(object, type = "lm")` and the number of terms, respectively.
- `all, which` If `all = FALSE` then `which` gives the integer index or a vector of logicals specifying the selection.
- `matrix.out` Logical. If `TRUE` then the constraint matrices are `cbind()`ed together. The result is usually more compact because the default is a list of constraint matrices.
- `colnames.arg, rownames.arg` Logical. If `TRUE` then column and row names are assigned corresponding to the variables.
- `...` Other possible arguments such as `type`.

Details

Constraint matrices describe the relationship of coefficients/component functions of a particular explanatory variable between the linear/additive predictors in VGLM/VGAM models. For example, they may be all different (constraint matrix is the identity matrix) or all the same (constraint matrix has one column and has unit values).

VGLMs and VGAMs have constraint matrices which are *known*. The class of RR-VGLMs have constraint matrices which are *unknown* and are to be estimated.

Value

The extractor function `constraints()` returns a list comprising of constraint matrices—usually one for each column of the VLM model matrix, and in that order. The list is labelled with the variable names. Each constraint matrix has M rows, where M is the number of linear/additive predictors, and whose rank is equal to the number of columns. A model with no constraints at all has an order M identity matrix as each variable's constraint matrix.

For vglm and vgam objects, feeding in type = "term" constraint matrices back into the same model should work and give an identical model. The default are the "lm"-type constraint matrices; this is a list with one constraint matrix per column of the LM matrix. See the constraints argument of vglm, and the example below.

Note

In all **VGAM** family functions zero = NULL means none of the linear/additive predictors are modelled as intercepts-only. Other arguments found in certain **VGAM** family functions which affect constraint matrices include parallel and exchangeable.

The constraints argument in vglm and vgam allows constraint matrices to be inputted. If so, then constraints(fit, type = "lm") can be fed into the constraints argument of the same object to get the same model.

The xij argument does not affect constraint matrices; rather, it allows each row of the constraint matrix to be multiplied by a specified vector.

Author(s)

T. W. Yee

References

Yee, T. W. and Wild, C. J. (1996). Vector generalized additive models. *Journal of the Royal Statistical Society, Series B, Methodological*, **58**, 481–493.

Yee, T. W. and Hastie, T. J. (2003). Reduced-rank vector generalized linear models. *Statistical Modelling*, **3**, 15–41.

See Also

is.parallel, is.zero, trim.constraints. VGLMs are described in vglm-class; RR-VGLMs are described in rrvglm-class.

Arguments such as zero and parallel found in many **VGAM** family functions are a way of creating/modifying constraint matrices conveniently, e.g., see zero. See CommonVGAMffArguments for more information.

Examples

```
# Fit the proportional odds model:
pneumo <- transform(pneumo, let = log(exposure.time))
(fit1 <- vglm(cbind(normal, mild, severe) ~ sm.bs(let, 3),
              cumulative(parallel = TRUE, reverse = TRUE), data = pneumo))
coef(fit1, matrix = TRUE)
constraints(fit1)  # Parallel assumption results in this
constraints(fit1, type = "term")  # Same as the default ("vlm"-type)
is.parallel(fit1)

# An equivalent model to fit1 (needs the type "term" constraints):
clist.term <- constraints(fit1, type = "term")  # "term"-type constraints
# cumulative() has no 'zero' argument to set to NULL (a good idea
```

```
# when using the 'constraints' argument):
(fit2 <- vglm(cbind(normal, mild, severe) ~ sm.bs(let, 3), data = pneumo,
              cumulative(reverse = TRUE), constraints = clist.term))
abs(max(coef(fit1, matrix = TRUE) -
        coef(fit2, matrix = TRUE)))  # Should be zero

# Fit a rank-1 stereotype (RR-multinomial logit) model:
fit <- rrvglm(Country ~ Width + Height + HP, multinomial, data = car.all)
constraints(fit)  # All except the first are the estimated A matrix
```

---

cops *Centre of the Parameter Space*

Description

Returns a vector similar to coefs() comprising the centre of the parameter space (COPS) values, given a fitted VGLM regression.

Usage

```
cops(object, ...)
copsvglm(object, beta.range = c(-5, 6),
         tol = .Machine$double.eps^0.25,
         dointercepts = TRUE, trace. = FALSE,
         slowtrain = FALSE, ...)
```

Arguments

`object`	A `vglm` object. However, this function will not work for all such objects.
`beta.range`	Numeric. Interval for the numerical search. After a little scaling, it is effectively fed into `interval` in `optimize`. Convergence failure may occur if this argument is too wide so it is a good idea to vary this argument. In fact, it is strongly recommended that several values be fed into this argument to help ensure the proper solution is obtained.
`tol`	Numeric. Fed into `tol` in `optimize`.
`dointercepts`	Logical. Compute the COPS for the intercepts? This should be set to `FALSE` for models such as `propodds` and `cumulative`.
`trace.`	Logical. Print a running log? This may or may not work properly.
`slowtrain`	Logical. If `TRUE` then all columns of a matrix is computed. If `FALSE` then only one column of a matrix is computed, and this is the only column needed.
`...`	currently unused but may be used in the future for further arguments passed into the other methods functions.

Details

For many models, some COPS values will be `Inf` or `-Inf` so that manual checking is needed, for example, `poissonff`. Each value returned may be effectively that of `beta.range` or `NA`. The answers returned by this function only make sense if the COPSs are in the interior of the parameter space. This function was written specifically for logistic regression but has much wider applicability. Currently the result returned depends critically on `beta.range` so that the answer should be checked after several values are fed into that argument.

Value

A named vector, similar to `coefvlm`. If `trace.` then a list is returned, having a componennt comprising a matrix of function evaluations used by `optimize`.

Note

This function is experimental and can be made to run more efficiently in the future.

Author(s)

Thomas W. Yee.

References

Yee, T. W. (2024). Musings and new results on the parameter space. *Under review*.

See Also

`hdeff`. `coefvlm`, `coef`.

Examples

```
## Not run: data("xs.nz", package = "VGAMdata")
data1 <- na.omit(xs.nz[, c("age", "cancer", "sex")])
fit1 <- vglm(cancer ~ age + sex, binomialff, data1)
cops(fit1)  # 'beta.range' is okay here

## End(Not run)
```

---

corbet *Corbet's Butterfly Data*

Description

About 3300 individual butterflies were caught in Malaya by naturalist Corbet trapping butterflies. They were classified to about 500 species.

Usage

```
data(corbet)
```

Format

A data frame with 24 observations on the following 2 variables.

`species` Number of species.

`ofreq` Observed frequency of individual butterflies of that species.

Details

In the early 1940s Corbet spent two years trapping butterflies in Malaya. Of interest was the total number of species. Some species were so rare (e.g., 118 species had only one specimen) that it was thought likely that there were many unknown species.

Actually, 119 species had over 24 observed frequencies, so this could/should be appended to the data set. Hence there are 620 species in total in a sample size of $n = 9031$ individuals.

References

Fisher, R. A., Corbet, A. S. and Williams, C. B. (1943). The Relation Between the Number of Species and the Number of Individuals in a Random Sample of an Animal Population. *Journal of Animal Ecology*, **12**, 42–58.

Examples

```
summary(corbet)
```

`cqo` *Fitting Constrained Quadratic Ordination (CQO)*

Description

A *constrained quadratic ordination* (CQO; formerly called *canonical Gaussian ordination* or CGO) model is fitted using the *quadratic reduced-rank vector generalized linear model* (QRR-VGLM) framework.

Usage

```
cqo(formula, family = stop("argument 'family' needs to be assigned"),
    data = list(), weights = NULL, subset = NULL,
    na.action = na.fail, etastart = NULL, mustart = NULL,
    coefstart = NULL, control = qrrvglm.control(...), offset = NULL,
    method = "cqo.fit", model = FALSE, x.arg = TRUE, y.arg = TRUE,
    contrasts = NULL, constraints = NULL, extra = NULL,
    smart = TRUE, ...)
```

Arguments

`formula` a symbolic description of the model to be fit. The RHS of the formula is applied to each linear predictor. Different variables in each linear predictor can be chosen by specifying constraint matrices.

`family` a function of class "vglmff" (see vglmff-class) describing what statistical model is to be fitted. This is called a "**VGAM** family function". See CommonVGAMffArguments for general information about many types of arguments found in this type of function. Currently the following families are supported: poissonff, binomialff (logitlink and clogloglink links available), negbinomial, gamma2. Sometimes special arguments are required for cqo(), e.g., binomialff(multiple.responses = TRUE).

`data` an optional data frame containing the variables in the model. By default the variables are taken from environment(formula), typically the environment from which cqo is called.

`weights` an optional vector or matrix of (prior) weights to be used in the fitting process. Currently, this argument should not be used.

`subset` an optional logical vector specifying a subset of observations to be used in the fitting process.

`na.action` a function which indicates what should happen when the data contain NAs. The default is set by the na.action setting of options, and is na.fail if that is unset. The "factory-fresh" default is na.omit.

`etastart` starting values for the linear predictors. It is a M-column matrix. If $M = 1$ then it may be a vector. Currently, this argument probably should not be used.

`mustart` starting values for the fitted values. It can be a vector or a matrix. Some family functions do not make use of this argument. Currently, this argument probably should not be used.

`coefstart` starting values for the coefficient vector. Currently, this argument probably should not be used.

`control` a list of parameters for controlling the fitting process. See qrrvglm.control for details.

`offset` This argument must not be used.

`method` the method to be used in fitting the model. The default (and presently only) method cqo.fit uses *iteratively reweighted least squares* (IRLS).

`model` a logical value indicating whether the *model frame* should be assigned in the model slot.

`x.arg, y.arg` logical values indicating whether the model matrix and response matrix used in the fitting process should be assigned in the x and y slots. Note the model matrix is the LM model matrix.

`contrasts` an optional list. See the contrasts.arg of model.matrix.default.

`constraints` an optional list of constraint matrices. The components of the list must be named with the term it corresponds to (and it must match in character format). Each constraint matrix must have M rows, and be of full-column rank. By default, constraint matrices are the M by M identity matrix unless arguments in the family function itself override these values. If `constraints` is used it must contain *all* the terms; an incomplete list is not accepted. Constraint matrices for x_2 variables are taken as the identity matrix.

`extra` an optional list with any extra information that might be needed by the family function.

`smart` logical value indicating whether smart prediction (`smartpred`) will be used.

`...` further arguments passed into `qrrvglm.control`.

Details

QRR-VGLMs or *constrained quadratic ordination* (CQO) models are estimated here by maximum likelihood estimation. Optimal linear combinations of the environmental variables are computed, called *latent variables* (these appear as `latvar` for $R = 1$ else `latvar1`, `latvar2`, etc. in the output). Here, R is the *rank* or the number of ordination axes. Each species' response is then a regression of these latent variables using quadratic polynomials on a transformed scale (e.g., log for Poisson counts, logit for presence/absence responses). The solution is obtained iteratively in order to maximize the log-likelihood function, or equivalently, minimize the deviance.

The central formula (for Poisson and binomial species data) is given by

$$\eta = B_1^T x_1 + A\nu + \sum_{m=1}^{M} (\nu^T D_m \nu) e_m$$

where x_1 is a vector (usually just a 1 for an intercept), x_2 is a vector of environmental variables, $\nu = C^T x_2$ is a R-vector of latent variables, e_m is a vector of 0s but with a 1 in the mth position. The η are a vector of linear/additive predictors, e.g., the mth element is $\eta_m = \log(E[Y_m])$ for the mth species. The matrices B_1, A, C and D_m are estimated from the data, i.e., contain the regression coefficients. The tolerance matrices satisfy $T_s = -\frac{1}{2} D_s^{-1}$. Many important CQO details are directly related to arguments in `qrrvglm.control`, e.g., the argument `noRRR` specifies which variables comprise x_1.

Theoretically, the four most popular **VGAM** family functions to be used with `cqo` correspond to the Poisson, binomial, normal, and negative binomial distributions. The latter is a 2-parameter model. All of these are implemented, as well as the 2-parameter gamma.

For initial values, the function `.Init.Poisson.QO` should work reasonably well if the data is Poisson with species having equal tolerances. It can be quite good on binary data too. Otherwise the `Cinit` argument in `qrrvglm.control` can be used.

It is possible to relax the quadratic form to an additive model. The result is a data-driven approach rather than a model-driven approach, so that CQO is extended to *constrained additive ordination* (CAO) when $R = 1$. See `cao` for more details.

In this documentation, M is the number of linear predictors, S is the number of responses (species). Then $M = S$ for Poisson and binomial species data, and $M = 2S$ for negative binomial and gamma distributed species data.

Incidentally, *Unconstrained quadratic ordination* (UQO) may be performed by, e.g., fitting a Goodman's RC association model; see `uqo` and the Yee and Hadi (2014) referenced there. For UQO, the response is the usual site-by-species matrix and there are no environmental variables; the site scores are free parameters. UQO can be performed under the assumption that all species have the same tolerance matrices.

Value

An object of class `"qrrvglm"`.

Warning

Local solutions are not uncommon when fitting CQO models. To increase the chances of obtaining the global solution, increase the value of the argument `Bestof` in `qrrvglm.control`. For reproducibility of the results, it pays to set a different random number seed before calling `cqo` (the function `set.seed` does this). The function `cqo` chooses initial values for **C** using `.Init.Poisson.QO()` if `Use.Init.Poisson.QO = TRUE`, else random numbers.

Unless `I.tolerances = TRUE` or `eq.tolerances = FALSE`, CQO is computationally expensive with memory and time. It pays to keep the rank down to 1 or 2. If `eq.tolerances = TRUE` and `I.tolerances = FALSE` then the cost grows quickly with the number of species and sites (in terms of memory requirements and time). The data needs to conform quite closely to the statistical model, and the environmental range of the data should be wide in order for the quadratics to fit the data well (bell-shaped response surfaces). If not, RR-VGLMs will be more appropriate because the response is linear on the transformed scale (e.g., log or logit) and the ordination is called *constrained linear ordination* or CLO.

Like many regression models, CQO is sensitive to outliers (in the environmental and species data), sparse data, high leverage points, multicollinearity etc. For these reasons, it is necessary to examine the data carefully for these features and take corrective action (e.g., omitting certain species, sites, environmental variables from the analysis, transforming certain environmental variables, etc.). Any optimum lying outside the convex hull of the site scores should not be trusted. Fitting a CAO is recommended first, then upon transformations etc., possibly a CQO can be fitted.

For binary data, it is necessary to have 'enough' data. In general, the number of sites n ought to be much larger than the number of species *S*, e.g., at least 100 sites for two species. Compared to count (Poisson) data, numerical problems occur more frequently with presence/absence (binary) data. For example, if `Rank = 1` and if the response data for each species is a string of all absences, then all presences, then all absences (when enumerated along the latent variable) then infinite parameter estimates will occur. In general, setting `I.tolerances = TRUE` may help.

This function was formerly called `cgo`. It has been renamed to reinforce a new nomenclature described in Yee (2006).

Note

The input requires care, preparation and thought—*a lot more* than other ordination methods. Here is a partial **checklist**.

(1) The number of species should be kept reasonably low, e.g., 12 max. Feeding in 100+ species wholesale is a recipe for failure. Choose a few species carefully. Using 10 well-chosen species is better than 100+ species thrown in willy-nilly.

(2) Each species should be screened individually first, e.g., for presence/absence is the species totally absent or totally present at all sites? For presence/absence data `sort(colMeans(data))` can help avoid such species.

(3) The number of explanatory variables should be kept low, e.g., 7 max.

(4) Each explanatory variable should be screened individually first, e.g., is it heavily skewed or are there outliers? They should be plotted and then transformed where needed. They should not be too highly correlated with each other.

(5) Each explanatory variable should be scaled, e.g., to mean 0 and unit variance. This is especially needed for `I.tolerance = TRUE`.

(6) Keep the rank low. Only if the data is very good should a rank-2 model be attempted. Usually a rank-1 model is all that is practically possible even after a lot of work. The rank-1 model should always be attempted first. Then might be clever and try use this for initial values for a rank-2 model.

(7) If the number of sites is large then choose a random sample of them. For example, choose a maximum of 500 sites. This will reduce the memory and time expense of the computations.

(8) Try `I.tolerance = TRUE` or `eq.tolerance = FALSE` if the inputted data set is large, so as to reduce the computational expense. That's because the default, `I.tolerance = FALSE` and `eq.tolerance = TRUE`, is very memory hungry.

By default, a rank-1 equal-tolerances QRR-VGLM model is fitted (see `qrrvglm.control` for the default control parameters). If `Rank > 1` then the latent variables are always transformed so that they are uncorrelated. By default, the argument `trace` is `TRUE` meaning a running log is printed out while the computations are taking place. This is because the algorithm is computationally expensive, therefore users might think that their computers have frozen if `trace = FALSE`!

The argument `Bestof` in `qrrvglm.control` controls the number of models fitted (each uses different starting values) to the data. This argument is important because convergence may be to a *local* solution rather than the *global* solution. Using more starting values increases the chances of finding the global solution. Always plot an ordination diagram (use the generic function `lvplot`) and see if it looks sensible. Local solutions arise because the optimization problem is highly nonlinear, and this is particularly true for CAO.

Many of the arguments applicable to `cqo` are common to `vglm` and `rrvglm.control`. The most important arguments are `Rank`, `noRRR`, `Bestof`, `I.tolerances`, `eq.tolerances`, `isd.latvar`, and `MUXfactor`.

When fitting a 2-parameter model such as the negative binomial or gamma, it pays to have `eq.tolerances = TRUE` and `I.tolerances = FALSE`. This is because numerical problems can occur when fitting the model far away from the global solution when `I.tolerances = TRUE`. Setting the two arguments as described will slow down the computation considerably, however it is numerically more stable.

In Example 1 below, an unequal-tolerances rank-1 QRR-VGLM is fitted to the hunting spiders dataset, and Example 2 is the equal-tolerances version. The latter is less likely to have convergence problems compared to the unequal-tolerances model. In Example 3 below, an equal-tolerances rank-2 QRR-VGLM is fitted to the hunting spiders dataset. The numerical difficulties encountered in fitting the rank-2 model suggests a rank-1 model is probably preferable. In Example 4 below, constrained binary quadratic ordination (in old nomenclature, constrained Gaussian logit ordination) is fitted to some simulated data coming from a species packing model. With multivariate binary responses, one must use `multiple.responses = TRUE` to indicate that the response (matrix) is multivariate. Otherwise, it is interpreted as a single binary response variable. In Example 5 below, the deviance residuals are plotted for each species. This is useful as a diagnostic plot. This is done by (re)regressing each species separately against the latent variable.

Sometime in the future, this function might handle input of the form `cqo(x, y)`, where `x` and `y` are matrices containing the environmental and species data respectively.

Author(s)

Thomas W. Yee. Thanks to Alvin Sou for converting a lot of the original FORTRAN code into C.

References

Yee, T. W. (2004). A new technique for maximum-likelihood canonical Gaussian ordination. *Ecological Monographs*, **74**, 685–701.

ter Braak, C. J. F. and Prentice, I. C. (1988). A theory of gradient analysis. *Advances in Ecological Research*, **18**, 271–317.

Yee, T. W. (2006). Constrained additive ordination. *Ecology*, **87**, 203–213.

See Also

qrrvglm.control, Coef.qrrvglm, predictqrrvglm, calibrate.qrrvglm, model.matrixqrrvglm, vcovqrrvglm, rcqo, cao, uqo, rrvglm, poissonff, binomialff, negbinomial, gamma2, lvplot.qrrvglm, perspqrrvglm, trplot.qrrvglm, vglm, set.seed, hspider, trapO.

Examples

```
## Not run:
# Example 1; Fit an unequal tolerances model to the hunting spiders data
hspider[,1:6] <- scale(hspider[,1:6])  # Standardized environmental variables
set.seed(1234)  # For reproducibility of the results
p1ut <- cqo(cbind(Alopacce, Alopcune, Alopfabr, Arctlute, Arctperi,
                  Auloalbi, Pardlugu, Pardmont, Pardnigr, Pardpull,
                  Trocterr, Zoraspin) ~
            WaterCon + BareSand + FallTwig + CoveMoss + CoveHerb + ReflLux,
            fam = poissonff, data = hspider, Crow1positive = FALSE,
            eq.tol = FALSE)
sort(deviance(p1ut, history = TRUE))  # A history of all the iterations
if (deviance(p1ut) > 1177) warning("suboptimal fit obtained")

S <- ncol(depvar(p1ut))  # Number of species
clr <- (1:(S+1))[-7]  # Omits yellow
lvplot(p1ut, y = TRUE, lcol = clr, pch = 1:S, pcol = clr,
       las = 1)  # Ordination diagram
legend("topright", leg = colnames(depvar(p1ut)), col = clr,
       pch = 1:S, merge = TRUE, bty = "n", lty = 1:S, lwd = 2)
(cp <- Coef(p1ut))

(a <- latvar(cp)[cp@latvar.order])  # Ordered site scores along the gradient
# Names of the ordered sites along the gradient:
rownames(latvar(cp))[cp@latvar.order]
(aa <- Opt(cp)[, cp@Optimum.order])  # Ordered optimums along the gradient
aa <- aa[!is.na(aa)]  # Delete the species that is not unimodal
names(aa)  # Names of the ordered optimums along the gradient

trplot(p1ut, which.species = 1:3, log = "xy", type = "b", lty = 1, lwd = 2,
       col = c("blue","red","green"), label = TRUE) -> ii  # Trajectory plot
legend(0.00005, 0.3, paste(ii$species[, 1], ii$species[, 2], sep = " and "),
       lwd = 2, lty = 1, col = c("blue", "red", "green"))
```

```
abline(a = 0, b = 1, lty = "dashed")

S <- ncol(depvar(p1ut))  # Number of species
clr <- (1:(S+1))[-7]  # Omits yellow
persp(p1ut, col = clr, label = TRUE, las = 1)  # Perspective plot


# Example 2; Fit an equal tolerances model. Less numerically fraught.
set.seed(1234)
p1et <- cqo(cbind(Alopacce, Alopcune, Alopfabr, Arctlute, Arctperi,
                  Auloalbi, Pardlugu, Pardmont, Pardnigr, Pardpull,
                  Trocterr, Zoraspin) ~
            WaterCon + BareSand + FallTwig + CoveMoss + CoveHerb + ReflLux,
            poissonff, data = hspider, Crow1positive = FALSE)
sort(deviance(p1et, history = TRUE))  # A history of all the iterations
if (deviance(p1et) > 1586) warning("suboptimal fit obtained")
S <- ncol(depvar(p1et))  # Number of species
clr <- (1:(S+1))[-7]  # Omits yellow
persp(p1et, col = clr, label = TRUE, las = 1)


# Example 3: A rank-2 equal tolerances CQO model with Poisson data
# This example is numerically fraught... need I.toler = TRUE too.
set.seed(555)
p2 <- cqo(cbind(Alopacce, Alopcune, Alopfabr, Arctlute, Arctperi,
                Auloalbi, Pardlugu, Pardmont, Pardnigr, Pardpull,
                Trocterr, Zoraspin) ~
          WaterCon + BareSand + FallTwig + CoveMoss + CoveHerb + ReflLux,
          poissonff, data = hspider, Crow1positive = FALSE,
          I.toler = TRUE, Rank = 2, Bestof = 3, isd.latvar = c(2.1, 0.9))
sort(deviance(p2, history = TRUE))  # A history of all the iterations
if (deviance(p2) > 1127) warning("suboptimal fit obtained")
lvplot(p2, ellips = FALSE, label = TRUE, xlim = c(-3,4),
       C = TRUE, Ccol = "brown", sites = TRUE, scol = "grey",
       pcol = "blue", pch = "+", chull = TRUE, ccol = "grey")


# Example 4: species packing model with presence/absence data
set.seed(2345)
n <- 200; p <- 5; S <- 5
mydata <- rcqo(n, p, S, fam = "binomial", hi.abundance = 4,
               eq.tol = TRUE, es.opt = TRUE, eq.max = TRUE)
myform <- attr(mydata, "formula")
set.seed(1234)
b1et <- cqo(myform, binomialff(multiple.responses = TRUE, link = "clogloglink"),
            data = mydata)
sort(deviance(b1et, history = TRUE))  # A history of all the iterations
lvplot(b1et, y = TRUE, lcol = 1:S, pch = 1:S, pcol = 1:S, las = 1)
Coef(b1et)

# Compare the fitted model with the 'truth'
cbind(truth = attr(mydata, "concoefficients"), fitted = concoef(b1et))
```

```
# Example 5: Plot the deviance residuals for diagnostic purposes
set.seed(1234)
p1et <- cqo(cbind(Alopacce, Alopcune, Alopfabr, Arctlute, Arctperi,
                  Auloalbi, Pardlugu, Pardmont, Pardnigr, Pardpull,
                  Trocterr, Zoraspin) ~
            WaterCon + BareSand + FallTwig + CoveMoss + CoveHerb + ReflLux,
            poissonff, data = hspider, eq.tol = TRUE, trace = FALSE)
sort(deviance(p1et, history = TRUE))  # A history of all the iterations
if (deviance(p1et) > 1586) warning("suboptimal fit obtained")
S <- ncol(depvar(p1et))
par(mfrow = c(3, 4))
for (ii in 1:S) {
  tempdata <- data.frame(latvar1 = c(latvar(p1et)),
                         sppCounts = depvar(p1et)[, ii])
  tempdata <- transform(tempdata, myOffset = -0.5 * latvar1^2)

# For species ii, refit the model to get the deviance residuals
  fit1 <- vglm(sppCounts ~ offset(myOffset) + latvar1, poissonff,
               data = tempdata, trace = FALSE)

# For checking: this should be 0
# print("max(abs(c(Coef(p1et)@B1[1,ii],Coef(p1et)@A[ii,1])-coef(fit1)))")
# print( max(abs(c(Coef(p1et)@B1[1,ii],Coef(p1et)@A[ii,1])-coef(fit1))) )

# Plot the deviance residuals
  devresid <- resid(fit1, type = "deviance")
  predvalues <- predict(fit1) + fit1@offset
  ooo <- with(tempdata, order(latvar1))
  plot(predvalues + devresid ~ latvar1, data = tempdata, col = "red",
       xlab = "latvar1", ylab = "", main = colnames(depvar(p1et))[ii])
  with(tempdata, lines(latvar1[ooo], predvalues[ooo], col = "blue"))
}

## End(Not run)
```

---

crashes *Crashes on New Zealand Roads in 2009*

Description

A variety of reported crash data cross-classified by time (hour of the day) and day of the week, accumulated over 2009. These include fatalities and injuries (by car), trucks, motor cycles, bicycles and pedestrians. There are some alcohol-related data too.

Usage

```
data(crashi)
data(crashf)
data(crashtr)
```

```
data(crashmc)
data(crashbc)
data(crashp)
data(alcoff)
data(alclevels)
```

Format

Data frames with hourly times as rows and days of the week as columns. The `alclevels` dataset has hourly times and alcohol levels.

Mon, Tue, Wed, Thu, Fri, Sat, Sun Day of the week.

0-30, 31-50, 51-80, 81-100, 101-120, 121-150, 151-200, 201-250, 251-300, 301-350, 350+ Blood alcohol level (milligrams alcohol per 100 millilitres of blood).

Details

Each cell is the aggregate number of crashes reported at each hour-day combination, over the 2009 calendar year. The `rownames` of each data frame is the start time (hourly from midnight onwards) on a 24 hour clock, e.g., 21 means 9.00pm to 9.59pm.

For crashes, `chrashi` are the number of injuries by car, `crashf` are the number of fatalities by car (not included in `chrashi`), `crashtr` are the number of crashes involving trucks, `crashmc` are the number of crashes involving motorcyclists, `crashbc` are the number of crashes involving bicycles, and `crashp` are the number of crashes involving pedestrians. For alcohol-related offences, `alcoff` are the number of alcohol offenders from breath screening drivers, and `alclevels` are the blood alcohol levels of fatally injured drivers.

Source

`http://www.transport.govt.nz/research/Pages/Motor-Vehicle-Crashes-in-New-Zealand-2009.aspx`. Thanks to Warwick Goold and Alfian F. Hadi for assistance.

References

Motor Vehicles Crashes in New Zealand 2009; Statistical Statement Calendar Year 2009. Ministry of Transport, NZ Government; Yearly Report 2010. ISSN: 1176-3949

See Also

`rrvglm`, `rcim`, `grc`.

Examples

```
## Not run:  plot(unlist(alcoff), type = "l", frame.plot = TRUE,
     axes = FALSE, col = "blue", bty = "o",
     main = "Alcoholic offenders on NZ roads, aggregated over 2009",
     sub  = "Vertical lines at midnight (purple) and noon (orange)",
     xlab = "Day/hour", ylab = "Number of offenders")
axis(1, at = 1 + (0:6) * 24 + 12, labels = colnames(alcoff))
axis(2, las = 1)
```

```
axis(3:4, labels = FALSE, tick = FALSE)
abline(v = sort(1 + c((0:7) * 24, (0:6) * 24 + 12)), lty = "dashed",
       col = c("purple", "orange"))
## End(Not run)

# Goodmans RC models
## Not run:
fitgrc1 <- grc(alcoff)  # Rank-1 model
fitgrc2 <- grc(alcoff, Rank = 2, Corner = FALSE, Uncor = TRUE)
Coef(fitgrc2)

## End(Not run)
## Not run:  biplot(fitgrc2, scaleA = 2.3, Ccol = "blue", Acol = "orange",
       Clabels = as.character(1:23), xlim = c(-1.3, 2.3),
       ylim = c(-1.2, 1))
## End(Not run)
```

`cratio` *Ordinal Regression with Continuation Ratios*

Description

Fits a continuation ratio logit/probit/cloglog/cauchit/... regression model to an ordered (preferably) factor response.

Usage

```
cratio(link = "logitlink", parallel = FALSE, reverse = FALSE,
    zero = NULL, ynames = FALSE, Thresh = NULL, Trev = reverse,
    Tref = if (Trev) "M" else 1, whitespace = FALSE)
```

Arguments

`link`	Link function applied to the M continuation ratio probabilities. See `Links` for more choices.
`parallel`	A logical, or formula specifying which terms have equal/unequal coefficients.
`reverse`	Logical. By default, the continuation ratios used are $\eta_j = logit(P[Y > j \| Y \geq j])$ for $j = 1, \ldots, M$. If `reverse` is `TRUE`, then $\eta_j = logit(P[Y < j+1 \| Y \leq j+1])$ will be used.
`ynames`	See `multinomial` for information.
`zero`	An integer-valued vector specifying which linear/additive predictors are modelled as intercepts only. The values must be from the set {1,2,...,M}. The default value means none are modelled as intercept-only terms. See `CommonVGAMffArguments` for more information.
`Thresh, Trev, Tref`	See `cumulative` for information. These arguments apply to ordinal categorical regression models.
`whitespace`	See `CommonVGAMffArguments` for information.

Details

In this help file the response Y is assumed to be a factor with ordered values $1, 2, \dots, M + 1$, so that M is the number of linear/additive predictors η_j.

There are a number of definitions for the *continuation ratio* in the literature. To make life easier, in the **VGAM** package, we use *continuation* ratios and *stopping* ratios (see `sratio`). Stopping ratios deal with quantities such as `logitlink(P[Y=j|Y>=j])`.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, `rrvglm` and `vgam`.

Warning

No check is made to verify that the response is ordinal if the response is a matrix; see `ordered`.

Boersch-Supan (2021) looks at sparse data and the numerical problems that result; see `sratio`.

Note

The response should be either a matrix of counts (with row sums that are all positive), or a factor. In both cases, the `y` slot returned by `vglm/vgam/rrvglm` is the matrix of counts.

For a nominal (unordered) factor response, the multinomial logit model (`multinomial`) is more appropriate.

Here is an example of the usage of the `parallel` argument. If there are covariates `x1`, `x2` and `x3`, then `parallel = TRUE ~ x1 + x2 -1` and `parallel = FALSE ~ x3` are equivalent. This would constrain the regression coefficients for `x1` and `x2` to be equal; those of the intercepts and `x3` would be different.

Author(s)

Thomas W. Yee

References

See `sratio`.

See Also

`sratio`, `acat`, `cumulative`, `multinomial`, `CM.equid`, `CommonVGAMffArguments`, `margeff`, `pneumo`, `budworm`, `logitlink`, `probitlink`, `clogloglink`, `cauchitlink`.

Examples

```
pneumo <- transform(pneumo, let = log(exposure.time))
(fit <- vglm(cbind(normal, mild, severe) ~ let,
             cratio(parallel = TRUE), data = pneumo))
coef(fit, matrix = TRUE)
constraints(fit)
predict(fit)
```

```
predict(fit, untransform = TRUE)
margeff(fit)
```

---

cumulative *Ordinal Regression with Cumulative Probabilities*

Description

Fits a cumulative link regression model to a (preferably ordered) factor response.

Usage

```
cumulative(link = "logitlink", parallel = FALSE,
    reverse = FALSE, multiple.responses = FALSE,
    ynames = FALSE, Thresh = NULL, Trev = reverse,
    Tref = if (Trev) "M" else 1, whitespace = FALSE)
```

Arguments

- `link` — Link function applied to the J cumulative probabilities. See `Links` for more choices, e.g., for the cumulative `probitlink`/`clogloglink`/... models.
- `parallel` — A logical or formula specifying which terms have equal/unequal coefficients. See below for more information about the parallelism assumption. The default results in what some people call the *generalized ordered logit model* to be fitted. If `parallel = TRUE` then it does not apply to the intercept.

  The *partial proportional odds model* can be fitted by assigning this argument something like `parallel = TRUE ~ -1 + x3 + x5` so that there is one regression coefficient for `x3` and `x5`. Equivalently, setting `parallel = FALSE ~ 1 + x2 + x4` means M regression coefficients for the intercept and `x2` and `x4`. It is important that the intercept is never parallel. See `CommonVGAMffArguments` for more information.
- `reverse` — Logical. By default, the cumulative probabilities used are $P(Y \leq 1)$, $P(Y \leq 2)$, ..., $P(Y \leq J)$. If `reverse` is `TRUE` then $P(Y \geq 2)$, $P(Y \geq 3)$, ..., $P(Y \geq J+1)$ are used.
- `ynames` — See `multinomial` for information.
- `multiple.responses` — Logical. Multiple responses? If `TRUE` then the input should be a matrix with values $1, 2, \ldots, L$, where $L = J + 1$ is the number of levels. Each column of the matrix is a response, i.e., multiple responses. A suitable matrix can be obtained from `Cut`.
- `Thresh` — Character. The choices concern constraint matrices applied to the intercepts. They can be constrained to be equally-spaced (*equid*istant) etc. See `CommonVGAMffArguments` and `constraints` for general information. Basically, the choice is pasted to the end of `"CM."` and that function is called. This means users can easily write their own `CM.`-type function.

If equally-spaced then the direction and the reference level are controlled by `Trev` and `Tref`, and the constraint matrix will be M by 2, with the second column corresponding to the distance between the thresholds.

If `"symm1"` then the fitted intercepts are *symmetric* about the median (M odd) intercept. If M is even then the median is the mean of the two most inner and adjacent intercepts. For this, `CM.symm1` is used to construct the appropriate constraint matrix.

If `"symm0"` then the median intercept is 0 by definition and the symmetry occurs about the origin. Thus the intercepts comprise pairs that differ by sign only. The appropriate constraint matrix is as with `"symm1"` but with the first column deleted. The choices `"symm1"` and `"symm0"` are effectively equivalent to `"symmetric"` and `"symmetric2"` respectively in **ordinal**.

For `"qnorm"` then `CM.qnorm` uses the `qnorm((1:M)/(M+1))` quantiles of the standard normal.

`Trev, Tref` Support arguments for `Thresh` for equally-spaced intercepts. The logical argument `Trev` is applied first to give the direction (i.e., ascending or descending) before row `Tref` (ultimately numeric) of the first (intercept) constraint matrix is set to the reference level. See `constraints` for information.

`whitespace` See `CommonVGAMffArguments` for information.

Details

In this help file the response Y is assumed to be a factor with ordered values $1, 2, \ldots, J+1$. Hence M is the number of linear/additive predictors η_j; for `cumulative()` one has $M = J$.

This **VGAM** family function fits the class of *cumulative link models* to (hopefully) an ordinal response. By default, the *non-parallel* cumulative logit model is fitted, i.e.,

$$\eta_j = logit(P[Y \leq j])$$

where $j = 1, 2, \ldots, M$ and the η_j are not constrained to be parallel. This is also known as the *non-proportional odds model*. If the logit link is replaced by a complementary log-log link (`clogloglink`) then this is known as the *proportional-hazards model*.

In almost all the literature, the constraint matrices associated with this family of models are known. For example, setting `parallel = TRUE` will make all constraint matrices (except for the intercept) equal to a vector of M 1's. If the constraint matrices are equal, unknown and to be estimated, then this can be achieved by fitting the model as a reduced-rank vector generalized linear model (RR-VGLM; see `rrvglm`). Currently, reduced-rank vector generalized additive models (RR-VGAMs) have not been implemented here.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Warning

No check is made to verify that the response is ordinal if the response is a matrix; see `ordered`.

Boersch-Supan (2021) looks at sparse data and the numerical problems that result; see `sratio`.

Note

The response should be either a matrix of counts (with row sums that are all positive), or a factor. In both cases, the y slot returned by vglm/vgam/rrvglm is the matrix of counts. The formula must contain an intercept term. Other **VGAM** family functions for an ordinal response include acat, cratio, sratio. For a nominal (unordered) factor response, the multinomial logit model (multinomial) is more appropriate.

With the logit link, setting parallel = TRUE will fit a proportional odds model. Note that the TRUE here does not apply to the intercept term. In practice, the validity of the *proportional odds assumption* needs to be checked, e.g., by a likelihood ratio test (LRT). If acceptable on the data, then numerical problems are less likely to occur during the fitting, and there are less parameters. Numerical problems occur when the linear/additive predictors cross, which results in probabilities outside of $(0, 1)$; setting parallel = TRUE will help avoid this problem.

Here is an example of the usage of the parallel argument. If there are covariates x2, x3 and x4, then parallel = TRUE ~ x2 + x3 -1 and parallel = FALSE ~ x4 are equivalent. This would constrain the regression coefficients for x2 and x3 to be equal; those of the intercepts and x4 would be different.

If the data is inputted in *long* format (not *wide* format, as in pneumo below) and the self-starting initial values are not good enough then try using mustart, coefstart and/or etatstart. See the example below.

To fit the proportional odds model one can use the **VGAM** family function propodds. Note that propodds(reverse) is equivalent to cumulative(parallel = TRUE, reverse = reverse) (which is equivalent to cumulative(parallel = TRUE, reverse = reverse, link = "logitlink")). It is for convenience only. A call to cumulative() is preferred since it reminds the user that a parallelism assumption is made, as well as being a lot more flexible.

Category specific effects may be modelled using the xij-facility; see vglm.control and fill1.

With most Threshold choices, the first few fitted regression coefficients need care in their interpretation. For example, some values could be the distance away from the median intercept. Typing something like constraints(fit)[[1]] gives the constraint matrix of the intercept term.

Author(s)

Thomas W. Yee

References

Agresti, A. (2013). *Categorical Data Analysis*, 3rd ed. Hoboken, NJ, USA: Wiley.

Agresti, A. (2010). *Analysis of Ordinal Categorical Data*, 2nd ed. Hoboken, NJ, USA: Wiley.

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*, 2nd ed. London: Chapman & Hall.

Tutz, G. (2012). *Regression for Categorical Data*, Cambridge: Cambridge University Press.

Tutz, G. and Berger, M. (2022). Sparser ordinal regression models based on parametric and additive location-shift approaches. *International Statistical Review*, **90**, 306–327. doi:10.1111/insr.12484.

Yee, T. W. (2010). The **VGAM** package for categorical data analysis. *Journal of Statistical Software*, **32**, 1–34. doi:10.18637/jss.v032.i10.

Yee, T. W. and Wild, C. J. (1996). Vector generalized additive models. *Journal of the Royal Statistical Society, Series B, Methodological*, **58**, 481–493.

See Also

propodds, constraints, CM.ones, CM.equid, R2latvar, ordsup, prplot, margeff, acat, cratio, sratio, multinomial, CommonVGAMffArguments, pneumo, budworm, Links, hdeff.vglm, logitlink, probitlink, clogloglink, cauchitlink, logistic1.

Examples

```
# Proportional odds model (p.179) of McCullagh and Nelder (1989)
pneumo <- transform(pneumo, let = log(exposure.time))
(fit <- vglm(cbind(normal, mild, severe) ~ let,
             cumulative(parallel = TRUE, reverse = TRUE), pneumo))
depvar(fit)  # Sample proportions (good technique)
fit@y        # Sample proportions (bad technique)
weights(fit, type = "prior")  # Number of observations
coef(fit, matrix = TRUE)
constraints(fit)  # Constraint matrices
apply(fitted(fit), 1, which.max)  # Classification
apply(predict(fit, newdata = pneumo, type = "response"),
      1, which.max)  # Classification
R2latvar(fit)

# Check that the model is linear in let ----------------------
fit2 <- vgam(cbind(normal, mild, severe) ~ s(let, df = 2),
             cumulative(reverse = TRUE), data = pneumo)
## Not run:
 plot(fit2, se = TRUE, overlay = TRUE, lcol = 1:2, scol = 1:2)
## End(Not run)

# Check the proportional odds assumption with a LRT ----------
(fit3 <- vglm(cbind(normal, mild, severe) ~ let,
              cumulative(parallel = FALSE, reverse = TRUE), pneumo))
pchisq(2 * (logLik(fit3) - logLik(fit)), df =
       length(coef(fit3)) - length(coef(fit)), lower.tail = FALSE)
lrtest(fit3, fit)  # More elegant

# A factor() version of fit ----------------------------------
# This is in long format (cf. wide format above)
Nobs <- round(depvar(fit) * c(weights(fit, type = "prior")))
sumNobs <- colSums(Nobs)  # apply(Nobs, 2, sum)

pneumo.long <-
  data.frame(symptoms = ordered(rep(rep(colnames(Nobs), nrow(Nobs)),
                                      times = c(t(Nobs))),
                              levels = colnames(Nobs)),
             let = rep(rep(with(pneumo, let), each = ncol(Nobs)),
                       times = c(t(Nobs))))
with(pneumo.long, table(let, symptoms))  # Should be same as pneumo
```

```
(fit.long1 <- vglm(symptoms ~ let, data = pneumo.long, trace = TRUE,
                   cumulative(parallel = TRUE, reverse = TRUE)))
coef(fit.long1, matrix = TRUE)  # cf. coef(fit, matrix = TRUE)
# Could try using mustart if fit.long1 failed to converge.
mymustart <- matrix(sumNobs / sum(sumNobs),
                    nrow(pneumo.long), ncol(Nobs), byrow = TRUE)
fit.long2 <- vglm(symptoms ~ let, mustart = mymustart,
                  cumulative(parallel = TRUE, reverse = TRUE),
                  data = pneumo.long, trace = TRUE)
coef(fit.long2, matrix = TRUE)  # cf. coef(fit, matrix = TRUE)
```

Dagum *The Dagum Distribution*

Description

Density, distribution function, quantile function and random generation for the Dagum distribution with shape parameters a and p, and scale parameter scale.

Usage

```
ddagum(x, scale = 1, shape1.a, shape2.p, log = FALSE)
pdagum(q, scale = 1, shape1.a, shape2.p, lower.tail = TRUE,
       log.p = FALSE)
qdagum(p, scale = 1, shape1.a, shape2.p, lower.tail = TRUE,
       log.p = FALSE)
rdagum(n, scale = 1, shape1.a, shape2.p)
```

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
n	number of observations. If length(n) > 1, the length is taken to be the number required.
shape1.a, shape2.p	shape parameters.
scale	scale parameter.
log	Logical. If log = TRUE then the logarithm of the density is returned.
lower.tail, log.p	Same meaning as in pnorm or qnorm.

Details

See dagum, which is the **VGAM** family function for estimating the parameters by maximum likelihood estimation.

Value

ddagum gives the density, pdagum gives the distribution function, qdagum gives the quantile function, and rdagum generates random deviates.

Note

The Dagum distribution is a special case of the 4-parameter generalized beta II distribution.

Author(s)

T. W. Yee and Kai Huang

References

Kleiber, C. and Kotz, S. (2003). *Statistical Size Distributions in Economics and Actuarial Sciences*, Hoboken, NJ, USA: Wiley-Interscience.

See Also

dagum, genbetaII.

Examples

```
probs <- seq(0.1, 0.9, by = 0.1)
shape1.a <- 1; shape2.p <- 2
# Should be 0:
max(abs(pdagum(qdagum(probs, shape1.a = shape1.a, shape2.p =
  shape2.p), shape1.a = shape1.a, shape2.p = shape2.p) - probs))

## Not run:  par(mfrow = c(1, 2))
x <- seq(-0.01, 5, len = 401)
plot(x, dexp(x), type = "l", col = "black",
     ylab = "", las = 1, ylim = c(0, 1),
     main = "Black is std exponential, others are ddagum(x, ...)")
lines(x, ddagum(x, shape1.a = shape1.a, shape2.p = 1), col = "orange")
lines(x, ddagum(x, shape1.a = shape1.a, shape2.p = 2), col = "blue")
lines(x, ddagum(x, shape1.a = shape1.a, shape2.p = 5), col = "green")
legend("topright", col = c("orange","blue","green"),
       lty = rep(1, len = 3), legend = paste("shape1.a =", shape1.a,
       ", shape2.p =", c(1, 2, 5)))

plot(x, pexp(x), type = "l", col = "black", ylab = "", las = 1,
     main = "Black is std exponential, others are pdagum(x, ...)")
lines(x, pdagum(x, shape1.a = shape1.a, shape2.p = 1), col = "orange")
lines(x, pdagum(x, shape1.a = shape1.a, shape2.p = 2), col = "blue")
lines(x, pdagum(x, shape1.a = shape1.a, shape2.p = 5), col = "green")
legend("bottomright", col = c("orange", "blue", "green"),
       lty = rep(1, len = 3), legend = paste("shape1.a =", shape1.a,
       ", shape2.p =", c(1, 2, 5)))

## End(Not run)
```

dagum *Dagum Distribution Family Function*

Description

Maximum likelihood estimation of the 3-parameter Dagum distribution.

Usage

```
dagum(lscale = "loglink", lshape1.a = "loglink", lshape2.p =
      "loglink", iscale = NULL, ishape1.a = NULL, ishape2.p =
      NULL, imethod = 1, lss = TRUE, gscale = exp(-5:5), gshape1.a
      = seq(0.75, 4, by = 0.25), gshape2.p = exp(-5:5), probs.y =
      c(0.25, 0.5, 0.75), zero = "shape")
```

Arguments

`lss` See `CommonVGAMffArguments` for important information.

`lshape1.a, lscale, lshape2.p`
Parameter link functions applied to the (positive) parameters `a`, `scale`, and `p`. See `Links` for more choices.

`iscale, ishape1.a, ishape2.p, imethod, zero`
See `CommonVGAMffArguments` for information. For `imethod = 2` a good initial value for `ishape2.p` is needed to obtain a good estimate for the other parameter.

`gscale, gshape1.a, gshape2.p`
See `CommonVGAMffArguments` for information.

`probs.y` See `CommonVGAMffArguments` for information.

Details

The 3-parameter Dagum distribution is the 4-parameter generalized beta II distribution with shape parameter $q = 1$. It is known under various other names, such as the Burr III, inverse Burr, beta-K, and 3-parameter kappa distribution. It can be considered a generalized log-logistic distribution. Some distributions which are special cases of the 3-parameter Dagum are the inverse Lomax ($a = 1$), Fisk ($p = 1$), and the inverse paralogistic ($a = p$). More details can be found in Kleiber and Kotz (2003).

The Dagum distribution has a cumulative distribution function

$$F(y) = [1 + (y/b)^{-a}]^{-p}$$

which leads to a probability density function

$$f(y) = apy^{ap-1}/[b^{ap}\{1 + (y/b)^a\}^{p+1}]$$

for $a > 0$, $b > 0$, $p > 0$, $y \geq 0$. Here, b is the scale parameter `scale`, and the others are shape parameters. The mean is

$$E(Y) = b\,\Gamma(p + 1/a)\,\Gamma(1 - 1/a)/\Gamma(p)$$

provided $-ap < 1 < a$; these are returned as the fitted values. This family function handles multiple responses.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Note

See the notes in genbetaII.

From Kleiber and Kotz (2003), the MLE is rather sensitive to isolated observations located sufficiently far from the majority of the data. Reliable estimation of the scale parameter require $n > 7000$, while estimates for a and p can be considered unbiased for $n > 2000$ or 3000.

Author(s)

T. W. Yee

References

Kleiber, C. and Kotz, S. (2003). *Statistical Size Distributions in Economics and Actuarial Sciences*, Hoboken, NJ, USA: Wiley-Interscience.

See Also

Dagum, genbetaII, betaII, sinmad, fisk, inv.lomax, lomax, paralogistic, inv.paralogistic, simulate.vlm.

Examples

```
## Not run:
ddata <- data.frame(y = rdagum(n = 3000, scale = exp(2),
                               shape1 = exp(1), shape2 = exp(1)))
fit <- vglm(y ~ 1, dagum(lss = FALSE), data = ddata, trace = TRUE)
fit <- vglm(y ~ 1, dagum(lss = FALSE, ishape1.a = exp(1)),
            data = ddata, trace = TRUE)
coef(fit, matrix = TRUE)
Coef(fit)
summary(fit)

## End(Not run)
```

---

dAR1 *The AR-1 Autoregressive Process*

Description

Density for the AR-1 model.

Usage

```
dAR1(x, drift = 0, var.error = 1, ARcoef1 = 0.0,
     type.likelihood = c("exact", "conditional"), log = FALSE)
```

Arguments

- `x,` vector of quantiles.
- `drift` the scaled mean (also known as the *drift* parameter), μ^*. Note that the mean is $\mu^*/(1-\rho)$. The default corresponds to observations that have mean 0.
- `log` Logical. If `TRUE` then the logarithm of the density is returned.
- `type.likelihood`, `var.error`, `ARcoef1` See `AR1`. The argument `ARcoef1` is ρ. The argument `var.error` is the variance of the i.i.d. random noise, i.e., σ^2. If `type.likelihood = "conditional"` then the first element or row of the result is currently assigned `NA`—this is because the density of the first observation is effectively ignored.

Details

Most of the background to this function is given in `AR1`. All the arguments are converted into matrices, and then all their dimensions are obtained. They are then coerced into the same size: the number of rows is the maximum of all the single rows, and ditto for the number of columns.

Value

`dAR1` gives the density.

Author(s)

T. W. Yee and Victor Miranda

See Also

`AR1`.

Examples

```
nn <- 100; set.seed(1)
tdata <- data.frame(index = 1:nn,
                    TS1 = arima.sim(nn, model = list(ar = -0.50),
                                    sd = exp(1)))
fit1 <- vglm(TS1 ~ 1, AR1, data = tdata, trace = TRUE)
rhobitlink(-0.5)
coef(fit1, matrix = TRUE)
(Cfit1 <- Coef(fit1))
summary(fit1)  # SEs are useful to know
logLik(fit1)
sum(dAR1(depvar(fit1), drift = Cfit1[1], var.error = (Cfit1[2])^2,
         ARcoef1 = Cfit1[3], log = TRUE))

fit2 <- vglm(TS1 ~ 1, AR1(type.likelihood = "cond"), data = tdata, trace = TRUE)
```

```
(Cfit2 <- Coef(fit2))  # Okay for intercept-only models
logLik(fit2)
head(keep <- dAR1(depvar(fit2), drift = Cfit2[1], var.error = (Cfit2[2])^2,
                  ARcoef1 = Cfit2[3], type.likelihood = "cond", log = TRUE))
sum(keep[-1])
```

---

deermice *Captures of Peromyscus maniculatus (Also Known as Deer Mice).*

Description

Captures of *Peromyscus maniculatus* collected at East Stuart Gulch, Colorado, USA.

Usage

```
data(deermice)
```

Format

The format is a data frame.

Details

Peromyscus maniculatus is a rodent native to North America. The deer mouse is small in size, only about 8 to 10 cm long, not counting the length of the tail.

Originally, the columns of this data frame represent the sex (`m` or `f`), the ages (`y`: young, `sa`: semi-adult, `a`: adult), the weights in grams, and the capture histories of 38 individuals over 6 trapping occasions (1: captured, 0: not captured).

The data set was collected by V. Reid and distributed with the **CAPTURE** program of Otis et al. (1978).

`deermice` has 38 deermice whereas `Perom` had 36 deermice (`Perom` has been withdrawn.) In `deermice` the two semi-adults have been classified as adults. The `sex` variable has 1 for female, and 0 for male.

References

Huggins, R. M. (1991). Some practical aspects of a conditional likelihood approach to capture experiments. *Biometrics*, **47**, 725–732.

Otis, D. L. et al. (1978). Statistical inference from capture data on closed animal populations, *Wildlife Monographs*, **62**, 3–135.

See Also

`posbernoulli.b`, `posbernoulli.t`, `fill1`.

Examples

```
head(deermice)
## Not run:
fit1 <- vglm(cbind(y1, y2, y3, y4, y5, y6) ~ sex + age,
             posbernoulli.t(parallel.t = TRUE), deermice, trace = TRUE)
coef(fit1)
coef(fit1, matrix = TRUE)

## End(Not run)
```

`deplot.lmscreg` *Density Plot for LMS Quantile Regression*

Description

Plots a probability density function associated with a LMS quantile regression.

Usage

```
deplot.lmscreg(object, newdata = NULL, x0, y.arg, show.plot =
                TRUE, ...)
```

Arguments

`object`	A **VGAM** quantile regression model, i.e., an object produced by modelling functions such as `vglm` and `vgam` with a family function beginning with `"lms."`, e.g., `lms.yjn`.
`newdata`	Optional data frame containing secondary variables such as sex. It should have a maximum of one row. The default is to use the original data.
`x0`	Numeric. The value of the primary variable at which to make the 'slice'.
`y.arg`	Numerical vector. The values of the response variable at which to evaluate the density. This should be a grid that is fine enough to ensure the plotted curves are smooth.
`show.plot`	Logical. Plot it? If `FALSE` no plot will be done.
`...`	Graphical parameter that are passed into `plotdeplot.lmscreg`.

Details

This function calls, e.g., `deplot.lms.yjn` in order to compute the density function.

Value

The original `object` but with a list placed in the slot `post`, called `@post$deplot`. The list has components

`newdata`	The argument `newdata` above, or a one-row data frame constructed out of the `x0` argument.
`y`	The argument `y.arg` above.
`density`	Vector of the density function values evaluated at `y.arg`.

Note

`plotdeplot.lmscreg` actually does the plotting.

Author(s)

Thomas W. Yee

References

Yee, T. W. (2004). Quantile regression via vector generalized additive models. *Statistics in Medicine*, **23**, 2295–2315.

See Also

`plotdeplot.lmscreg`, `qtplot.lmscreg`, `lms.bcn`, `lms.bcg`, `lms.yjn`.

Examples

```
## Not run:
fit <- vgam(BMI ~ s(age, df = c(4, 2)), lms.bcn(zero = 1), bmi.nz)
ygrid <- seq(15, 43, by = 0.25)
deplot(fit, x0 = 20, y = ygrid, xlab = "BMI", col = "green", llwd = 2,
    main = "BMI distribution at ages 20 (green), 40 (blue), 60 (red)")
deplot(fit, x0 = 40, y = ygrid, add = TRUE, col = "blue", llwd = 2)
deplot(fit, x0 = 60, y = ygrid, add = TRUE, col = "red", llwd = 2) -> a

names(a@post$deplot)
a@post$deplot$newdata
head(a@post$deplot$y)
head(a@post$deplot$density)

## End(Not run)
```

depvar *Response Variable Extracted*

Description

A generic function that extracts the response/dependent variable from objects.

Usage

```
depvar(object, ...)
```

Arguments

`object`	An object that has some response/dependent variable.
`...`	Other arguments fed into the specific methods function of the model. In particular, sometimes `type = c("lm", "lm2")` is available, in which case the first one is chosen if the user does not input a value. The latter value corresponds to argument `form2`, and sometimes a response for that is optional.

Details

By default this function is preferred to calling `fit@y`, say.

Value

The response/dependent variable, usually as a matrix or vector.

Author(s)

Thomas W. Yee

See Also

`model.matrix`, `vglm`.

Examples

```
pneumo <- transform(pneumo, let = log(exposure.time))
(fit <- vglm(cbind(normal, mild, severe) ~ let, propodds, pneumo))
fit@y        # Sample proportions (not recommended)
depvar(fit)  # Better than using fit@y
weights(fit, type = "prior")  # Number of observations
```

dextlogF *Extended log-F Distribution*

Description

Density for the extended log-F distribution.

Usage

```
dextlogF(x, lambda, tau, location = 0, scale = 1, log = FALSE)
```

Arguments

x	Vector of quantiles.
lambda, tau	See extlogF1.
location, scale	See extlogF1.
log	If TRUE then the log density is returned, else the density.

Details

The details are given in extlogF1.

Value

dextlogF gives the density.

Author(s)

T. W. Yee

See Also

extlogF1, dalap.

Examples

```
## Not run:  x <- seq(-2, 8, by = 0.1); mytau <- 0.25; mylambda <- 0.2
plot(x, dextlogF(x, mylambda, tau = mytau), type = "l",
     las = 1, col = "blue", ylab = "PDF (log-scale)", log = "y",
     main = "Extended log-F density function is blue",
     sub = "Asymmetric Laplace is orange dashed")
lines(x, dalap(x, tau = mytau, scale = 3.5), col = "orange", lty = 2)
abline(v = 0, col = "gray", lty = 2)
## End(Not run)
```

`df.residual` *Residual Degrees-of-Freedom*

Description

Returns the residual degrees-of-freedom extracted from a fitted VGLM object.

Usage

```
df.residual_vlm(object, type = c("vlm", "lm"), ...)
```

Arguments

`object`	an object for which the degrees-of-freedom are desired, e.g., a `vglm` object.
`type`	the type of residual degrees-of-freedom wanted. In some applications the 'usual' LM-type value may be more appropriate. The default is the first choice.
`...`	additional optional arguments.

Details

When a VGLM is fitted, a *large* (VLM) generalized least squares (GLS) fit is done at each IRLS iteration. To do this, an ordinary least squares (OLS) fit is performed by transforming the GLS using Cholesky factors. The number of rows is M times the 'ordinary' number of rows of the LM-type model: nM. Here, M is the number of linear/additive predictors. So the formula for the VLM-type residual degrees-of-freedom is $nM - p^*$ where p^* is the number of columns of the 'big' VLM matrix. The formula for the LM-type residual degrees-of-freedom is $n - p_j$ where p_j is the number of columns of the 'ordinary' LM matrix corresponding to the jth linear/additive predictor.

Value

The value of the residual degrees-of-freedom extracted from the object. When `type = "vlm"` this is a single integer, and when `type = "lm"` this is a M-vector of integers.

See Also

`vglm`, `deviance`, `lm`, `anova.vglm`,

Examples

```
pneumo <- transform(pneumo, let = log(exposure.time))
(fit <- vglm(cbind(normal, mild, severe) ~ let, propodds, pneumo))
head(model.matrix(fit, type = "vlm"))
head(model.matrix(fit, type = "lm"))

df.residual(fit, type = "vlm")  # n * M - p_VLM
nobs(fit, type = "vlm")  # n * M
nvar(fit, type = "vlm")  # p_VLM
```

```
df.residual(fit, type = "lm")  # n - p_LM(j)
nobs(fit, type = "lm")  # n
nvar(fit, type = "lm")  # p_LM
nvar_vlm(fit, type = "lm")  # p_LM(j) (<= p_LM elementwise)
```

---

dgaitdplot *Plotting the GAITD Combo Density*

Description

Plots a 1- or 2-parameter GAITD combo probability mass function.

Usage

```
dgaitdplot(theta.p, fam = "pois", a.mix = NULL, i.mix = NULL,
    d.mix = NULL, a.mlm = NULL, i.mlm = NULL,
    d.mlm = NULL, truncate = NULL, max.support = Inf,
    pobs.mix = 0, pobs.mlm = 0,
    pstr.mix = 0, pstr.mlm = 0,
    pdip.mix = 0, pdip.mlm = 0, byrow.aid = FALSE,
    theta.a = theta.p, theta.i = theta.p, theta.d = theta.p,
    deflation = FALSE, plot.it = TRUE, new.plot = TRUE,
    offset.x = ifelse(new.plot, 0, 0.25), type.plot = "h",
    xlim = c(0, min(100, max.support + 2)),
    ylim = NULL, xlab = "", ylab = "Probability", main = "",
    cex.main = 1.2, posn.main = NULL,
    all.col = NULL, all.lty = NULL, all.lwd = NULL,
    lty.p = "solid", lty.a.mix = "longdash", lty.a.mlm = "longdash",
    lty.i.mix = "dashed", lty.i.mlm = "dashed",
    lty.d.mix = "solid", lty.d.mlm = "solid", lty.d.dip = "dashed",
    col.p = "pink2",
    col.a.mix = artichoke.col, col.a.mlm = asparagus.col,
    col.i.mix = indigo.col, col.i.mlm = iris.col,
    col.d.mix = deer.col, col.d.mlm = dirt.col, col.d.dip = desire.col,
    col.t = turquoise.col, cex.p = 1, lwd.p = NULL, lwd.a = NULL,
    lwd.i = NULL, lwd.d = NULL, iontop = TRUE, dontop = TRUE,
    las = 0, lend = "round", axes.x = TRUE, axes.y = TRUE,
    Plot.trunc = TRUE, cex.t = 1, pch.t = 1,
    baseparams.argnames = NULL, nparams = 1, flip.args = FALSE, ...)
```

Arguments

theta.p	Numeric, usually scalar but may have length 2. This matches with, e.g., `lambda.p` for `Gaitdpois`. A length 2 example is `c(size.p, munb.p)` for `Gaitdnbinom`, in which case `fam = "nbinom"`. Another length 2 example is `c(mean.p, dispind.p)` for `Gaitgenpois1`, in which case `fam = "genpois1"`.
fam	Character, `paste0("dgait", fam)` should be a d-type function returning the PMF. The default is for the GAITD Poisson combo.

a.mix, i.mix, a.mlm, i.mlm
: See Gaitdpois and gaitdpoisson.

d.mix, d.mlm
: See Gaitdpois and gaitdpoisson.

truncate, max.support
: See Gaitdpois and gaitdpoisson.

pobs.mix, pobs.mlm, byrow.aid
: See Gaitdpois and gaitdpoisson.

pstr.mix, pstr.mlm, pdip.mix, pdip.mlm
: See Gaitdpois and gaitdpoisson.

theta.a, theta.i, theta.d
: Similar to theta.p, and they should have the same length too.

deflation
: Logical. Plot the deflation (dip) probabilities?

plot.it
: Logical. Plot the PMF?

new.plot, offset.x
: If new.plot then plot is called. If multiple plots are desired then use offset.x to shift the lines.

xlim, ylim, xlab, ylab
: See par and plot.default. Argument xlim should be integer-valued.

main, cex.main, posn.main
: Character, size and position of main for the title. See title, par and plot.default. The position is used if it is a 2-vector.

all.col, all.lty, all.lwd
: These arguments allow all the colours, line types and line widths arguments to be assigned to these values, i.e., so that they are the same for all values of the support. For example, if all.lwd = 2 then this sets lwd.p, lwd.a, lwd.i and lwd.d all equal to 2.

lty.p, lty.a.mix, lty.a.mlm, lty.i.mix, lty.i.mlm
: Line type for parent, altered and inflated. See par and plot.default.

col.p, col.a.mix, col.a.mlm, col.i.mix, col.i.mlm
: Line colour for parent (nonspecial), altered, inflated, truncated and deflated values. See par and plot.default. Roughly, by default and currently, the parent is pink-like, the altered are greenish, the inflated are purplish/violet, the truncated are light blue, and the deflated are brownish with the dip probabilities being reddish. The proper colour names are similar to being acrostic. For each operator, the colours of "mix" vs "mlm" are similar but different—this is intentional. Warning: the default colours might change, depending on style!

lty.d.mix, lty.d.mlm, lty.d.dip
: Similar to above. Used when deflation = TRUE.

col.d.mix, col.d.mlm, col.d.dip
: Similar to above. Used when deflation = TRUE. The website https://www.spycolor.com was used to choose some of the default colours; the first two are also called "dirt" and "deer" respectively, which are both brownish.

col.t
: Point colour for truncated values, the default is "tan".

type.plot, cex.p
: The former matches 'type' argument in plot.default. The latter is the size of the point if type.plot = "p" or type.plot = "b", etc.

lwd.p, lwd.a, lwd.i, lwd.d
: Line width for parent, altered and inflated. See par and plot.default. By default par()\$lwd is used for all of them.

las, lend
: See par.

iontop, dontop
: Logicals. Draw the inflated and deflated bars on top? The default is to draw the spikes on top, but if FALSE then the spikes are drawn from the bottom—this makes it easier to see their distribution. Likewise, if deflation = TRUE then dontop is used to position the deflation (dip) probabilities.

axes.x, axes.y
: Logical. Plot axes? See par and plot.default.

Plot.trunc, cex.t, pch.t
: Logical. Plot the truncated values? If so, then specify the size and plotting character. See par and plot.default.

baseparams.argnames
: Character string specifying the argument name for the generic parameter theta, e.g., "lambda" for gaitdpoisson, By appending .p, there is an argument called lambda.p in dgaitdpois. Another example is for gaitdlog: "shape" appended with .p means that dgaitdlog should have an argument called shape.p. This argument is optional and increases the reliability of the do.call call internally.

nparams, flip.args
: Not for use by the user. It is used internally to handle the NBD.

...
: Currently unused but there is provision for passing graphical arguments in in the future; see par.

Details

This is meant to be a crude function to plot the PMF of the GAITD combo model. Some flexibility is offered via many graphical arguments, but there are still many improvements that could be done.

Value

A list is returned invisibly. The components are:

x
: The integer values between the values of xlim.

pmf.z
: The value of the PMF, by calling the d-type function with all the arguments fed in.

sc.parent
: The same level as the scaled parent distribution. Thus for inflated values, the value where the spikes begin. And for deflated values, the value at the top of the dips. This is a convenient way to obtain them as it is quite cumbersome to compute them manually. For any nonspecial value, such as non-inflated and non-deflated values, they are equal to pmf.z.

unsc.parent
: Unscaled parent distribution. If there is no alteration, inflation, deflation and truncation then this is the basic PMF stipulated by the parent distribution only. Usually this is FYI only.

Note

This utility function may change a lot in the future. Because this function is called by a **shiny** app, if any parameter values lie outside the parameter space then `stop` will be called. For example, too much deflation results in `NaN` values returned by `dgaitdnbinom`.

Author(s)

T. W. Yee.

See Also

`plotdgaitd`, `spikeplot`, `meangaitd`, `Gaitdpois`, `gaitdpoisson`, `Gaitdnbinom`, `multilogitlink`.

Examples

```
## Not run:   i.mix <- seq(0, 25, by = 5)
mean.p <- 10; size.p <- 8
dgaitdplot(c(size.p, mean.p), fam = "nbinom", xlim = c(0, 25),
     a.mix = i.mix + 1, i.mix = i.mix, pobs.mix = 0.1,
     pstr.mix = 0.1, lwd.i = 2,lwd.p = 2, lwd.a = 2)

## End(Not run)
```

`dhuber` *Huber's Least Favourable Distribution*

Description

Density, distribution function, quantile function and random generation for Huber's least favourable distribution, see Huber and Ronchetti (2009).

Usage

```
 dhuber(x, k = 0.862, mu = 0, sigma = 1, log = FALSE)
edhuber(x, k = 0.862, mu = 0, sigma = 1, log = FALSE)
 rhuber(n, k = 0.862, mu = 0, sigma = 1)
 qhuber(p, k = 0.862, mu = 0, sigma = 1, lower.tail = TRUE,
        log.p = FALSE)
 phuber(q, k = 0.862, mu = 0, sigma = 1, lower.tail = TRUE,
        log.p = FALSE)
```

Arguments

`x, q`	numeric vector, vector of quantiles.
`p`	vector of probabilities.
`n`	number of random values to be generated. If `length(n)` > 1 then the length is taken to be the number required.

k	numeric. Borderline value of central Gaussian part of the distribution. This is known as the tuning constant, and should be positive. For example, k = 0.862 refers to a 20% contamination neighborhood of the Gaussian distribution. If k = 1.40 then this is 5% contamination.
mu	numeric. distribution mean.
sigma	numeric. Distribution scale (sigma = 1 defines the distribution in standard form, with standard Gaussian centre).
log	Logical. If log = TRUE then the logarithm of the result is returned.
lower.tail, log.p	Same meaning as in pnorm or qnorm.

Details

Details are given in huber2, the **VGAM** family function for estimating the parameters mu and sigma.

Value

dhuber gives out a vector of density values.

edhuber gives out a list with components val (density values) and eps (contamination proportion).

rhuber gives out a vector of random numbers generated by Huber's least favourable distribution.

phuber gives the distribution function, qhuber gives the quantile function.

Author(s)

Christian Hennig wrote [d,ed,r]huber() (from **smoothmest**) and slight modifications were made by T. W. Yee to replace looping by vectorization and addition of the log argument. Arash Ardalan wrote [pq]huber(), and two arguments for these were implemented by Kai Huang. This helpfile was adapted from **smoothmest**.

See Also

huber2.

Examples

```
set.seed(123456)
edhuber(1:5, k = 1.5)
rhuber(5)

## Not run:  mu <- 3; xx <- seq(-2, 7, len = 100)  # Plot CDF and PDF
plot(xx, dhuber(xx, mu = mu), type = "l", col = "blue", las = 1,
     main = "blue is density, orange is the CDF", ylab = "",
     sub = "Purple lines are the 10,20,...,90 percentiles",
     ylim = 0:1)
abline(h = 0, col = "blue", lty = 2)
lines(xx, phuber(xx, mu = mu), type = "l", col = "orange")
probs <- seq(0.1, 0.9, by = 0.1)
Q <- qhuber(probs, mu = mu)
```

```
lines(Q, dhuber(Q, mu = mu), col = "purple", lty = 3, type = "h")
lines(Q, phuber(Q, mu = mu), col = "purple", lty = 3, type = "h")
abline(h = probs, col = "purple", lty = 3)
phuber(Q, mu = mu) - probs  # Should be all 0s

## End(Not run)
```

Diffzeta *Differenced Zeta Distribution*

Description

Density, distribution function, quantile function, and random generation for the differenced zeta distribution.

Usage

```
ddiffzeta(x, shape, start = 1, log = FALSE)
pdiffzeta(q, shape, start = 1, lower.tail = TRUE)
qdiffzeta(p, shape, start = 1)
rdiffzeta(n, shape, start = 1)
```

Arguments

`x, q, p, n`	Same as in `runif`.
`shape, start`	Details at `diffzeta`.
`log, lower.tail`	Same as in `runif`.

Details

This distribution appears to work well on the distribution of English words in such texts. Some more details are given in `diffzeta`.

Value

`ddiffzeta` gives the density, `pdiffzeta` gives the distribution function, `qdiffzeta` gives the quantile function, and `rdiffzeta` generates random deviates.

Note

Given some response data, the **VGAM** family function `diffzeta` estimates the parameter `shape`.

Function `pdiffzeta()` suffers from the problems that `plog` sometimes has, i.e., when `p` is very close to 1.

Author(s)

T. W. Yee

See Also

diffzeta, zetaff, zipf, Oizeta.

Examples

```
ddiffzeta(1:20, 0.5, start = 2)
rdiffzeta(20, 0.5)

## Not run:  shape <- 0.8; x <- 1:10
plot(x, ddiffzeta(x, sh = shape), type = "h", ylim = 0:1, las = 1,
     sub = "shape=0.8", col = "blue", ylab = "Probability",
     main = "Differenced zeta distribution: blue=PMF; orange=CDF")
lines(x + 0.1, pdiffzeta(x, shape = shape), col = "orange",
      lty = 3, type = "h")
## End(Not run)
```

---

diffzeta *Differenced Zeta Distribution Family Function*

Description

Estimates the parameter of the differenced zeta distribution.

Usage

```
diffzeta(start = 1, lshape = "loglink", ishape = NULL)
```

Arguments

lshape, ishape	Same as zetaff.
start	Smallest value of the support of the distribution. Must be a positive integer.

Details

The PMF is

$$P(Y = y) = (a/y)^s - (a/(1 + y))^s, \;\; s > 0, \;\; y = a, a + 1, \ldots,$$

where s is the positive shape parameter, and a is `start`. According to Moreno-Sanchez et al. (2016), this model fits quite well to about 40 percent of all the English books in the Project Gutenberg data base (about 30,000 texts). Multiple responses are handled.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Author(s)

T. W. Yee

References

Moreno-Sanchez, I., Font-Clos, F. and Corral, A. (2016). Large-Scale Analysis of Zipf's Law in English Texts, *PLoS ONE*, **11**(1), 1–19.

See Also

Diffzeta, zetaff, zeta, zipf, zipf.

Examples

```
odata <- data.frame(x2 = runif(nn <- 1000))  # Artificial data
odata <- transform(odata, shape = loglink(-0.25 + x2, inv = TRUE))
odata <- transform(odata, y1 = rdiffzeta(nn, shape))
with(odata, table(y1))
ofit <- vglm(y1 ~ x2, diffzeta, odata, trace = TRUE)
coef(ofit, matrix = TRUE)
```

---

dirichlet *Fitting a Dirichlet Distribution*

Description

Fits a Dirichlet distribution to a matrix of compositions.

Usage

```
dirichlet(link = "loglink", parallel = FALSE, zero = NULL,
          imethod = 1)
```

Arguments

`link` Link function applied to each of the M (positive) shape parameters α_j. See Links for more choices. The default gives $\eta_j = \log(\alpha_j)$.

`parallel, zero, imethod` See CommonVGAMffArguments for more information.

Details

In this help file the response is assumed to be a M-column matrix with positive values and whose rows each sum to unity. Such data can be thought of as compositional data. There are M linear/additive predictors η_j.

The Dirichlet distribution is commonly used to model compositional data, including applications in genetics. Suppose $(Y_1, \ldots, Y_M)^T$ is the response. Then it has a Dirichlet distribution if $(Y_1, \ldots, Y_{M-1})^T$ has density

$$\frac{\Gamma(\alpha_+)}{\prod_{j=1}^M \Gamma(\alpha_j)} \prod_{j=1}^M y_j^{\alpha_j - 1}$$

where $\alpha_+ = \alpha_1 + \cdots + \alpha_M$, $\alpha_j > 0$, and the density is defined on the unit simplex

$$\Delta_M = \left\{ (y_1, \ldots, y_M)^T : y_1 > 0, \ldots, y_M > 0, \sum_{j=1}^M y_j = 1 \right\}.$$

One has $E(Y_j) = \alpha_j / \alpha_+$, which are returned as the fitted values. For this distribution Fisher scoring corresponds to Newton-Raphson.

The Dirichlet distribution can be motivated by considering the random variables $(G_1, \ldots, G_M)^T$ which are each independent and identically distributed as a gamma distribution with density $f(g_j) = g_j^{\alpha_j - 1} e^{-g_j} / \Gamma(\alpha_j)$. Then the Dirichlet distribution arises when $Y_j = G_j / (G_1 + \cdots + G_M)$.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, `rrvglm` and `vgam`.

When fitted, the `fitted.values` slot of the object contains the M-column matrix of means.

Note

The response should be a matrix of positive values whose rows each sum to unity. Similar to this is count data, where probably a multinomial logit model (`multinomial`) may be appropriate. Another similar distribution to the Dirichlet is the Dirichlet-multinomial (see `dirmultinomial`).

Author(s)

Thomas W. Yee

References

Lange, K. (2002). *Mathematical and Statistical Methods for Genetic Analysis*, 2nd ed. New York: Springer-Verlag.

Forbes, C., Evans, M., Hastings, N. and Peacock, B. (2011). *Statistical Distributions*, Hoboken, NJ, USA: John Wiley and Sons, Fourth edition.

See Also

`rdiric`, `dirmultinomial`, `multinomial`, `simplex`.

Examples

```
ddata <- data.frame(rdiric(1000,
                    shape = exp(c(y1 = -1, y2 = 1, y3 = 0))))
fit <- vglm(cbind(y1, y2, y3)  ~ 1, dirichlet,
            data = ddata, trace = TRUE, crit = "coef")
Coef(fit)
coef(fit, matrix = TRUE)
head(fitted(fit))
```

---

dirmul.old *Fitting a Dirichlet-Multinomial Distribution*

Description

Fits a Dirichlet-multinomial distribution to a matrix of non-negative integers.

Usage

```
dirmul.old(link = "loglink", ialpha = 0.01, parallel = FALSE,
           zero = NULL)
```

Arguments

link	Link function applied to each of the M (positive) shape parameters α_j for $j = 1, \ldots, M$. See Links for more choices. Here, M is the number of columns of the response matrix.
ialpha	Numeric vector. Initial values for the alpha vector. Must be positive. Recycled to length M.
parallel	A logical, or formula specifying which terms have equal/unequal coefficients.
zero	An integer-valued vector specifying which linear/additive predictors are modelled as intercepts only. The values must be from the set {1,2,…,M}. See CommonVGAMffArguments for more information.

Details

The Dirichlet-multinomial distribution, which is somewhat similar to a Dirichlet distribution, has probability function

$$P(Y_1 = y_1, \ldots, Y_M = y_M) = \binom{2y_*}{y_1, \ldots, y_M} \frac{\Gamma(\alpha_+)}{\Gamma(2y_* + \alpha_+)} \prod_{j=1}^{M} \frac{\Gamma(y_j + \alpha_j)}{\Gamma(\alpha_j)}$$

for $\alpha_j > 0$, $\alpha_+ = \alpha_1 + \cdots + \alpha_M$, and $2y_* = y_1 + \cdots + y_M$. Here, $\binom{a}{b}$ means "a choose b" and refers to combinations (see choose). The (posterior) mean is

$$E(Y_j) = (y_j + \alpha_j)/(2y_* + \alpha_+)$$

for $j = 1, \ldots, M$, and these are returned as the fitted values as a M-column matrix.

Value

An object of class "vglmff" (see `vglmff-class`). The object is used by modelling functions such as `vglm`, `rrvglm` and `vgam`.

Note

The response should be a matrix of non-negative values. Convergence seems to slow down if there are zero values. Currently, initial values can be improved upon.

This function is almost defunct and may be withdrawn soon. Use `dirmultinomial` instead.

Author(s)

Thomas W. Yee

References

Lange, K. (2002). *Mathematical and Statistical Methods for Genetic Analysis*, 2nd ed. New York: Springer-Verlag.

Forbes, C., Evans, M., Hastings, N. and Peacock, B. (2011). *Statistical Distributions*, Hoboken, NJ, USA: John Wiley and Sons, Fourth edition.

Paul, S. R., Balasooriya, U. and Banerjee, T. (2005). Fisher information matrix of the Dirichlet-multinomial distribution. *Biometrical Journal*, **47**, 230–236.

Tvedebrink, T. (2010). Overdispersion in allelic counts and θ-correction in forensic genetics. *Theoretical Population Biology*, **78**, 200–210.

See Also

`dirmultinomial`, `dirichlet`, `betabinomialff`, `multinomial`.

Examples

```
# Data from p.50 of Lange (2002)
alleleCounts <- c(2, 84, 59, 41, 53, 131, 2, 0,
       0, 50, 137, 78, 54, 51, 0, 0,
       0, 80, 128, 26, 55, 95, 0, 0,
       0, 16, 40, 8, 68, 14, 7, 1)
dim(alleleCounts) <- c(8, 4)
alleleCounts <- data.frame(t(alleleCounts))
dimnames(alleleCounts) <- list(c("White","Black","Chicano","Asian"),
                    paste("Allele", 5:12, sep = ""))

set.seed(123)  # @initialize uses random numbers
fit <- vglm(cbind(Allele5,Allele6,Allele7,Allele8,Allele9,
                  Allele10,Allele11,Allele12) ~ 1, dirmul.old,
             trace = TRUE, crit = "c", data = alleleCounts)

(sfit <- summary(fit))
vcov(sfit)
round(eta2theta(coef(fit),
```

```
                fit@misc$link,
                fit@misc$earg), digits = 2)  # not preferred
round(Coef(fit), digits = 2)  # preferred
round(t(fitted(fit)), digits = 4)  # 2nd row of Lange (2002, Table 3.5)
coef(fit, matrix = TRUE)


pfit <- vglm(cbind(Allele5,Allele6,Allele7,Allele8,Allele9,
                   Allele10,Allele11,Allele12) ~ 1,
             dirmul.old(parallel = TRUE), trace = TRUE,
             data = alleleCounts)
round(eta2theta(coef(pfit, matrix = TRUE), pfit@misc$link,
                pfit@misc$earg), digits = 2)  # 'Right' answer
round(Coef(pfit), digits = 2)  # 'Wrong' due to parallelism constraint
```

`dirmultinomial` *Fitting a Dirichlet-Multinomial Distribution*

Description

Fits a Dirichlet-multinomial distribution to a matrix response.

Usage

```
dirmultinomial(lphi = "logitlink", iphi = 0.10, parallel = FALSE,
               zero = "M")
```

Arguments

`lphi`	Link function applied to the ϕ parameter, which lies in the open unit interval $(0, 1)$. See `Links` for more choices.
`iphi`	Numeric. Initial value for ϕ. Must be in the open unit interval $(0, 1)$. If a failure to converge occurs then try assigning this argument a different value.
`parallel`	A logical (formula not allowed here) indicating whether the probabilities $\pi_1, \ldots, \pi_{M-1}$ are to be equal via equal coefficients. Note π_M will generally be different from the other probabilities. Setting `parallel = TRUE` will only work if you also set `zero = NULL` because of interference between these arguments (with respect to the intercept term).
`zero`	An integer-valued vector specifying which linear/additive predictors are modelled as intercepts only. The values must be from the set $\{1, 2, \ldots, M\}$. If the character `"M"` then this means the numerical value M, which corresponds to linear/additive predictor associated with ϕ. Setting `zero = NULL` means none of the values from the set $\{1, 2, \ldots, M\}$. See `CommonVGAMffArguments` for more information.

Details

The Dirichlet-multinomial distribution arises from a multinomial distribution where the probability parameters are not constant but are generated from a multivariate distribution called the Dirichlet distribution. The Dirichlet-multinomial distribution has probability function

$$P(Y_1 = y_1, \ldots, Y_M = y_M) = \binom{N_*}{y_1, \ldots, y_M} \frac{\prod_{j=1}^{M} \prod_{r=1}^{y_j} (\pi_j (1 - \phi) + (r - 1)\phi)}{\prod_{r=1}^{N_*} (1 - \phi + (r - 1)\phi)}$$

where ϕ is the *over-dispersion* parameter and $N_* = y_1 + \cdots + y_M$. Here, $\binom{a}{b}$ means "a choose b" and refers to combinations (see choose). The above formula applies to each row of the matrix response. In this **VGAM** family function the first $M - 1$ linear/additive predictors correspond to the first $M - 1$ probabilities via

$$\eta_j = \log(P[Y = j]/P[Y = M]) = \log(\pi_j/\pi_M)$$

where η_j is the jth linear/additive predictor ($\eta_M = 0$ by definition for $P[Y = M]$ but not for ϕ) and $j = 1, \ldots, M - 1$. The Mth linear/additive predictor corresponds to `lphi` applied to ϕ.

Note that $E(Y_j) = N_* \pi_j$ but the probabilities (returned as the fitted values) π_j are bundled together as a M-column matrix. The quantities N_* are returned as the prior weights.

The beta-binomial distribution is a special case of the Dirichlet-multinomial distribution when $M = 2$; see betabinomial. It is easy to show that the first shape parameter of the beta distribution is $shape1 = \pi(1/\phi - 1)$ and the second shape parameter is $shape2 = (1 - \pi)(1/\phi - 1)$. Also, $\phi = 1/(1 + shape1 + shape2)$, which is known as the *intra-cluster correlation* coefficient.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, rrvglm and vgam.

If the model is an intercept-only model then `@misc` (which is a list) has a component called `shape` which is a vector with the M values $\pi_j(1/\phi - 1)$.

Warning

This **VGAM** family function is prone to numerical problems, especially when there are covariates.

Note

The response can be a matrix of non-negative integers, or else a matrix of sample proportions and the total number of counts in each row specified using the `weights` argument. This dual input option is similar to multinomial.

To fit a 'parallel' model with the ϕ parameter being an intercept-only you will need to use the `constraints` argument.

Currently, Fisher scoring is implemented. To compute the expected information matrix a `for` loop is used; this may be very slow when the counts are large. Additionally, convergence may be slower than usual due to round-off error when computing the expected information matrices.

Author(s)

Thomas W. Yee

References

Paul, S. R., Balasooriya, U. and Banerjee, T. (2005). Fisher information matrix of the Dirichlet-multinomial distribution. *Biometrical Journal*, **47**, 230–236.

Tvedebrink, T. (2010). Overdispersion in allelic counts and θ-correction in forensic genetics. *Theoretical Population Biology*, **78**, 200–210.

Yu, P. and Shaw, C. A. (2014). An Efficient Algorithm for Accurate Computation of the Dirichlet-Multinomial Log-Likelihood Function. *Bioinformatics*, **30**, 1547–54.

See Also

`dirmul.old`, `betabinomial`, `betabinomialff`, `dirichlet`, `multinomial`.

Examples

```
nn <- 5; M <- 4; set.seed(1)
ydata <- data.frame(round(matrix(runif(nn * M, max = 100), nn, M)))
colnames(ydata) <- paste("y", 1:M, sep = "")  # Integer counts

fit <- vglm(cbind(y1, y2, y3, y4) ~ 1, dirmultinomial,
            data = ydata, trace = TRUE)
head(fitted(fit))
depvar(fit)  # Sample proportions
weights(fit, type = "prior", matrix = FALSE)  # Total counts per row

## Not run:
ydata <- transform(ydata, x2 = runif(nn))
fit <- vglm(cbind(y1, y2, y3, y4) ~ x2, dirmultinomial,
            data = ydata, trace = TRUE)
Coef(fit)
coef(fit, matrix = TRUE)
(sfit <- summary(fit))
vcov(sfit)

## End(Not run)
```

`dlogF` *log F Distribution*

Description

Density for the log F distribution.

Usage

```
dlogF(x, shape1, shape2, log = FALSE)
```

Arguments

`x`	Vector of quantiles.
`shape1, shape2`	Positive shape parameters.
`log`	if `TRUE` then the log density is returned, else the density.

Details

The details are given in `logF`.

Value

`dlogF` gives the density.

Author(s)

T. W. Yee

See Also

`hypersecant`, `dextlogF`.

Examples

```
## Not run:  shape1 <- 1.5; shape2 <- 0.5; x <- seq(-5, 8, length = 1001)
plot(x, dlogF(x, shape1, shape2), type = "l",
     las = 1, col = "blue", ylab = "pdf",
     main = "log F density function")

## End(Not run)
```

`double.cens.normal` *Univariate Normal Distribution with Double Censoring*

Description

Maximum likelihood estimation of the two parameters of a univariate normal distribution when there is double censoring.

Usage

```
double.cens.normal(r1 = 0, r2 = 0, lmu = "identitylink", lsd =
       "loglink", imu = NULL, isd = NULL, zero = "sd")
```

Arguments

`r1, r2`	Integers. Number of smallest and largest values censored, respectively.
`lmu, lsd`	Parameter link functions applied to the mean and standard deviation. See `Links` for more choices.
`imu, isd, zero`	See `CommonVGAMffArguments` for more information.

Details

This family function uses the Fisher information matrix given in Harter and Moore (1966). The matrix is not diagonal if either `r1` or `r2` are positive.

By default, the mean is the first linear/additive predictor and the log of the standard deviation is the second linear/additive predictor.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Note

This family function only handles a vector or one-column matrix response. The `weights` argument, if used, are interpreted as frequencies, therefore it must be a vector with positive integer values.

With no censoring at all (the default), it is better (and equivalent) to use `uninormal`.

Author(s)

T. W. Yee

References

Harter, H. L. and Moore, A. H. (1966). Iterative maximum-likelihood estimation of the parameters of normal populations from singly and doubly censored samples. *Biometrika*, **53**, 205–213.

See Also

`uninormal`, `cens.normal`, `tobit`.

Examples

```
## Not run:  # Repeat the simulations of Harter & Moore (1966)
SIMS <- 100  # Number of simulations (change this to 1000)
mu.save <- sd.save <- rep(NA, len = SIMS)
r1 <- 0; r2 <- 4; nn <- 20
for (sim in 1:SIMS) {
  y <- sort(rnorm(nn))
  y <- y[(1+r1):(nn-r2)]  # Delete r1 smallest and r2 largest
  fit <- vglm(y ~ 1, double.cens.normal(r1 = r1, r2 = r2))
  mu.save[sim] <- predict(fit)[1, 1]
  sd.save[sim] <- exp(predict(fit)[1, 2])  # Assumes a log link & ~ 1
}
c(mean(mu.save), mean(sd.save))  # Should be c(0,1)
c(sd(mu.save), sd(sd.save))

## End(Not run)

# Data from Sarhan & Greenberg (1962); MLEs are mu=9.2606, sd=1.3754
strontium90 <- data.frame(y = c(8.2, 8.4, 9.1, 9.8, 9.9))
```

```
fit <- vglm(y ~ 1, double.cens.normal(r1 = 2, r2 = 3, isd = 6),
            data = strontium90, trace = TRUE)
coef(fit, matrix = TRUE)
Coef(fit)
```

`double.expbinomial` *Double Exponential Binomial Distribution Family Function*

Description

Fits a double exponential binomial distribution by maximum likelihood estimation. The two parameters here are the mean and dispersion parameter.

Usage

```
double.expbinomial(lmean = "logitlink", ldispersion = "logitlink",
                   idispersion = 0.25, zero = "dispersion")
```

Arguments

`lmean, ldispersion`
: Link functions applied to the two parameters, called μ and θ respectively below. See `Links` for more choices. The defaults cause the parameters to be restricted to $(0, 1)$.

`idispersion`
: Initial value for the dispersion parameter. If given, it must be in range, and is recyled to the necessary length. Use this argument if convergence failure occurs.

`zero`
: A vector specifying which linear/additive predictor is to be modelled as intercept-only. If assigned, the single value can be either `1` or `2`. The default is to have a single dispersion parameter value. To model both parameters as functions of the covariates assign `zero = NULL`. See `CommonVGAMffArguments` for more details.

Details

This distribution provides a way for handling overdispersion in a binary response. The double exponential binomial distribution belongs the family of double exponential distributions proposed by Efron (1986). Below, equation numbers refer to that original article. Briefly, the idea is that an ordinary one-parameter exponential family allows the addition of a second parameter θ which varies the dispersion of the family without changing the mean. The extended family behaves like the original family with sample size changed from n to $n\theta$. The extended family is an exponential family in μ when n and θ are fixed, and an exponential family in θ when n and μ are fixed. Having $0 < \theta < 1$ corresponds to overdispersion with respect to the binomial distribution. See Efron (1986) for full details.

This **VGAM** family function implements an *approximation* (2.10) to the exact density (2.4). It replaces the normalizing constant by unity since the true value nearly equals 1. The default model fitted is $\eta_1 = logit(\mu)$ and $\eta_2 = logit(\theta)$. This restricts both parameters to lie between 0 and 1, although the dispersion parameter can be modelled over a larger parameter space by assigning the arguments `ldispersion` and `edispersion`.

Approximately, the mean (of Y) is μ. The *effective sample size* is the dispersion parameter multiplied by the original sample size, i.e., $n\theta$. This family function uses Fisher scoring, and the two estimates are asymptotically independent because the expected information matrix is diagonal.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm.

Warning

Numerical difficulties can occur; if so, try using idispersion.

Note

This function processes the input in the same way as binomialff, however multiple responses are not allowed (binomialff(multiple.responses = FALSE)).

Author(s)

T. W. Yee

References

Efron, B. (1986). Double exponential families and their use in generalized linear regression. *Journal of the American Statistical Association*, **81**, 709–721.

See Also

binomialff, toxop, CommonVGAMffArguments.

Examples

```
# This example mimics the example in Efron (1986).
# The results here differ slightly.

# Scale the variables
toxop <- transform(toxop,
                   phat = positive / ssize,
                   srainfall = scale(rainfall),  # (6.1)
                   sN = scale(ssize))            # (6.2)

# A fit similar (should be identical) to Sec.6 of Efron (1986).
# But does not use poly(), and M = 1.25 here, as in (5.3)
cmlist <- list("(Intercept)"    = diag(2),
               "I(srainfall)"   = rbind(1, 0),
               "I(srainfall^2)" = rbind(1, 0),
               "I(srainfall^3)" = rbind(1, 0),
               "I(sN)" = rbind(0, 1),
               "I(sN^2)" = rbind(0, 1))
fit <-
  vglm(cbind(phat, 1 - phat) * ssize ~
```

```
       I(srainfall) + I(srainfall^2) + I(srainfall^3) +
       I(sN) + I(sN^2),
       double.expbinomial(ldisp = extlogitlink(min = 0, max = 1.25),
                          idisp = 0.2, zero = NULL),
       toxop, trace = TRUE, constraints = cmlist)

# Now look at the results
coef(fit, matrix = TRUE)
head(fitted(fit))
summary(fit)
vcov(fit)
sqrt(diag(vcov(fit)))  # Standard errors

# Effective sample size (not quite the last column of Table 1)
head(predict(fit))
Dispersion <- extlogitlink(predict(fit)[,2], min = 0, max = 1.25,
                           inverse = TRUE)
c(round(weights(fit, type = "prior") * Dispersion, digits = 1))


# Ordinary logistic regression (gives same results as (6.5))
ofit <- vglm(cbind(phat, 1 - phat) * ssize ~
             I(srainfall) + I(srainfall^2) + I(srainfall^3),
             binomialff, toxop, trace = TRUE)


# Same as fit but it uses poly(), and can be plotted (cf. Fig.1)
cmlist2 <- list("(Intercept)"                 = diag(2),
                "poly(srainfall, degree = 3)" = rbind(1, 0),
                "poly(sN, degree = 2)"        = rbind(0, 1))
fit2 <-
  vglm(cbind(phat, 1 - phat) * ssize ~
       poly(srainfall, degree = 3) + poly(sN, degree = 2),
       double.expbinomial(ldisp = extlogitlink(min = 0, max = 1.25),
                          idisp = 0.2, zero = NULL),
       toxop, trace = TRUE, constraints = cmlist2)
## Not run:  par(mfrow = c(1, 2))  # Cf. Fig.1
plot(as(fit2, "vgam"), se = TRUE, lcol = "blue", scol = "orange")

# Cf. Figure 1(a)
par(mfrow = c(1,2))
ooo <- with(toxop, sort.list(rainfall))
with(toxop, plot(rainfall[ooo], fitted(fit2)[ooo], type = "l",
                 col = "blue", las = 1, ylim = c(0.3, 0.65)))
with(toxop, points(rainfall[ooo], fitted(ofit)[ooo],
                   col = "orange", type = "b", pch = 19))

# Cf. Figure 1(b)
ooo <- with(toxop, sort.list(ssize))
with(toxop, plot(ssize[ooo], Dispersion[ooo], type = "l",
                 col = "blue", las = 1, xlim = c(0, 100)))
## End(Not run)
```

`ducklings` *Relative Frequencies of Serum Proteins in White Pekin Ducklings*

Description

Relative frequencies of serum proteins in white Pekin ducklings as determined by electrophoresis.

Usage

```
data(ducklings)
```

Format

The format is: chr "ducklings"

Details

Columns `p1`, `p2`, `p3` stand for pre-albumin, albumin, globulins respectively. These were collected from 3-week old white Pekin ducklings. Let Y_1 be proportional to the total milligrams of pre-albumin in the blood serum of a duckling. Similarly, let Y_2 and Y_3 be directly proportional to the same factor as Y_1 to the total milligrams respectively of albumin and globulins in its blood serum. The proportion of pre-albumin is given by $Y_1/(Y_1 + Y_2 + Y_3)$, and similarly for the others.

Source

Mosimann, J. E. (1962) On the compound multinomial distribution, the multivariate β-distribution, and correlations among proportions, Biometrika, **49**, 65–82.

See Also

`dirichlet`.

Examples

```
print(ducklings)
```

`eCDF` *Empirical Cumulative Distribution Function*

Description

Returns the desired quantiles of quantile regression object such as an extlogF1() or lms.bcn() VGLM object

Usage

```
eCDF.vglm(object, all = FALSE, ...)
```

Arguments

`object`	an object such as a `vglm` object with family function `extlogF1` or `lms.bcn`.
`all`	Logical. Return all other information? If true, the empirical CDF is returned.
`...`	additional optional arguments. Currently unused.

Details

This function was specifically written for a `vglm` object with family function `extlogF1` or `lms.bcn`. It returns the proportion of data lying below each of the fitted quantiles, and optionally the desired quantiles (arguments `tau` or `percentiles / 100` in the family function). The output is coerced to be comparable between family functions by calling the columns by the same names.

Value

A vector with each value lying in (0, 1). If `all = TRUE` then a 2-column matrix with the second column being the `tau` values or equivalent.

See Also

`extlogF1`, `lms.bcn`, `vglm`.

Examples

```
fit1 <- vglm(BMI ~ ns(age, 4), extlogF1, data = bmi.nz)  # trace = TRUE
eCDF(fit1)
eCDF(fit1, all = TRUE)
```

---

enzyme — *Enzyme Data*

Description

Enzyme velocity and substrate concentration.

Usage

```
data(enzyme)
```

Format

A data frame with 12 observations on the following 2 variables.

conc a numeric explanatory vector; substrate concentration

velocity a numeric response vector; enzyme velocity

Details

Sorry, more details need to be included later.

Source

Sorry, more details need to be included later.

References

Watts, D. G. (1981). An introduction to nonlinear least squares. In: L. Endrenyi (Ed.), *Kinetic Data Analysis: Design and Analysis of Enzyme and Pharmacokinetic Experiments*, pp.1–24. New York: Plenum Press.

See Also

micmen.

Examples

```
## Not run:
fit <- vglm(velocity ~ 1, micmen, data = enzyme, trace = TRUE,
            form2 = ~ conc - 1, crit = "crit")
summary(fit)

## End(Not run)
```

---

erf — *Error Function, and variants*

Description

Computes the error function, or its inverse, based on the normal distribution. Also computes the complement of the error function, or its inverse,

Usage

```
erf(x, inverse = FALSE)
erfc(x, inverse = FALSE)
```

Arguments

x	Numeric.
inverse	Logical. Of length 1.

Details

$Erf(x)$ is defined as

$$Erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x \exp(-t^2) dt$$

so that it is closely related to pnorm. The inverse function is defined for x in $(-1, 1)$.

Value

Returns the value of the function evaluated at `x`.

Note

Some authors omit the term $2/\sqrt{\pi}$ from the definition of $Erf(x)$. Although defined for complex arguments, this function only works for real arguments.

The *complementary error function* $erfc(x)$ is defined as $1 - erf(x)$, and is implemented by `erfc`. Its inverse function is defined for x in $(0, 2)$.

Author(s)

T. W. Yee

References

Abramowitz, M. and Stegun, I. A. (1972). *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, New York: Dover Publications Inc.

See Also

`pnorm`.

Examples

```
## Not run:
curve(erf,   -3, 3, col = "orange", ylab = "", las = 1)
curve(pnorm, -3, 3, add = TRUE, col = "blue", lty = "dotted", lwd = 2)
abline(v = 0, h = 0, lty = "dashed")
legend("topleft", c("erf(x)", "pnorm(x)"), col = c("orange", "blue"),
       lty = c("solid", "dotted"), lwd = 1:2)
## End(Not run)
```

`erlang` *Erlang Distribution*

Description

Estimates the scale parameter of the Erlang distribution by maximum likelihood estimation.

Usage

```
erlang(shape.arg, lscale = "loglink", imethod = 1, zero = NULL)
```

Arguments

shape.arg
: The shape parameters. The user must specify a positive integer, or integers for multiple responses. They are recycled `by.row = TRUE` according to `matrix`.

lscale
: Link function applied to the (positive) $scale$ parameter. See `Links` for more choices.

imethod, zero
: See `CommonVGAMffArguments` for more details.

Details

The Erlang distribution is a special case of the gamma distribution with *shape* that is a positive integer. If `shape.arg = 1` then it simplifies to the exponential distribution. As illustrated in the example below, the Erlang distribution is the distribution of the sum of `shape.arg` independent and identically distributed exponential random variates.

The probability density function of the Erlang distribution is given by

$$f(y) = \exp(-y/scale) y^{shape-1} scale^{-shape} / \Gamma(shape)$$

for known positive integer $shape$, unknown $scale > 0$ and $y > 0$. Here, $\Gamma(shape)$ is the gamma function, as in `gamma`. The mean of *Y* is $\mu = shape \times scale$ and its variance is $shape \times scale^2$. The linear/additive predictor, by default, is $\eta = \log(scale)$.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Note

Multiple responses are permitted. The `rate` parameter found in `gammaR` is `1/scale` here—see also `rgamma`.

Author(s)

T. W. Yee

References

Most standard texts on statistical distributions describe this distribution, e.g.,

Forbes, C., Evans, M., Hastings, N. and Peacock, B. (2011). *Statistical Distributions*, Hoboken, NJ, USA: John Wiley and Sons, Fourth edition.

See Also

`gammaR`, `exponential`, `simulate.vlm`.

Examples

```
rate <- exp(2); myshape <- 3
edata <- data.frame(y = rep(0, nn <- 1000))
for (ii in 1:myshape)
  edata <- transform(edata, y = y + rexp(nn, rate = rate))
fit <- vglm(y ~ 1, erlang(shape = myshape), edata, trace = TRUE)
coef(fit, matrix = TRUE)
Coef(fit)  # Answer = 1/rate
1/rate
summary(fit)
```

`Expectiles-Exponential` *Expectiles of the Exponential Distribution*

Description

Density function, distribution function, and expectile function and random generation for the distribution associated with the expectiles of an exponential distribution.

Usage

```
deexp(x, rate = 1, log = FALSE)
peexp(q, rate = 1, lower.tail = TRUE, log.p = FALSE)
qeexp(p, rate = 1, Maxit.nr = 10, Tol.nr = 1.0e-6,
      lower.tail = TRUE, log.p = FALSE)
reexp(n, rate = 1)
```

Arguments

- `x, p, q` See `deunif`.
- `n, rate, log` See `rexp`.
- `lower.tail, log.p` Same meaning as in `pexp` or `qexp`.
- `Maxit.nr, Tol.nr` See `deunif`.

Details

General details are given in `deunif` including a note regarding the terminology used. Here, `exp` corresponds to the distribution of interest, F, and `eexp` corresponds to G. The addition of "e" is for the 'other' distribution associated with the parent distribution. Thus `deexp` is for g, `peexp` is for G, `qeexp` is for the inverse of G, `reexp` generates random variates from g.

For `qeexp` the Newton-Raphson algorithm is used to solve for y satisfying $p = G(y)$. Numerical problems may occur when values of `p` are very close to 0 or 1.

Value

deexp(x) gives the density function $g(x)$. peexp(q) gives the distribution function $G(q)$. qeexp(p) gives the expectile function: the value y such that $G(y) = p$. reexp(n) gives n random variates from G.

Author(s)

T. W. Yee and Kai Huang

See Also

deunif, denorm, dexp.

Examples

```
my.p <- 0.25; y <- rexp(nn <- 1000)
(myexp <- qeexp(my.p))
sum(myexp - y[y <= myexp]) / sum(abs(myexp - y))  # Should be my.p

## Not run:  par(mfrow = c(2,1))
yy <- seq(-0, 4, len = nn)
plot(yy, deexp(yy),  col = "blue", ylim = 0:1, xlab = "y", ylab = "g(y)",
     type = "l", main = "g(y) for Exp(1); dotted green is f(y) = dexp(y)")
lines(yy, dexp(yy), col = "green", lty = "dotted", lwd = 2)  # 'original'

plot(yy, peexp(yy), type = "l", col = "blue", ylim = 0:1,
     xlab = "y", ylab = "G(y)", main = "G(y) for Exp(1)")
abline(v = 1, h = 0.5, col = "red", lty = "dashed")
lines(yy, pexp(yy), col = "green", lty = "dotted", lwd = 2)
## End(Not run)
```

Expectiles-Normal — *Expectiles of the Normal Distribution*

Description

Density function, distribution function, and expectile function and random generation for the distribution associated with the expectiles of a normal distribution.

Usage

```
denorm(x, mean = 0, sd = 1, log = FALSE)
penorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qenorm(p, mean = 0, sd = 1, Maxit.nr = 10, Tol.nr = 1.0e-6,
       lower.tail = TRUE, log.p = FALSE)
renorm(n, mean = 0, sd = 1)
```

Arguments

x, p, q — See deunif.

n, mean, sd, log — See rnorm.

lower.tail, log.p — Same meaning as in pnorm or qnorm.

Maxit.nr, Tol.nr — See deunif.

Details

General details are given in deunif including a note regarding the terminology used. Here, norm corresponds to the distribution of interest, F, and enorm corresponds to G. The addition of "e" is for the 'other' distribution associated with the parent distribution. Thus denorm is for g, penorm is for G, qenorm is for the inverse of G, renorm generates random variates from g.

For qenorm the Newton-Raphson algorithm is used to solve for y satisfying $p = G(y)$. Numerical problems may occur when values of p are very close to 0 or 1.

Value

denorm(x) gives the density function $g(x)$. penorm(q) gives the distribution function $G(q)$. qenorm(p) gives the expectile function: the value y such that $G(y) = p$. renorm(n) gives n random variates from G.

Author(s)

T. W. Yee and Kai Huang

See Also

deunif, deexp, dnorm, amlnormal, lms.bcn.

Examples

```
my.p <- 0.25; y <- rnorm(nn <- 1000)
(myexp <- qenorm(my.p))
sum(myexp - y[y <= myexp]) / sum(abs(myexp - y))  # Should be my.p

# Non-standard normal
mymean <- 1; mysd <- 2
yy <- rnorm(nn, mymean, mysd)
(myexp <- qenorm(my.p, mymean, mysd))
sum(myexp - yy[yy <= myexp]) / sum(abs(myexp - yy))  # Should be my.p
penorm(-Inf, mymean, mysd)      #  Should be 0
penorm( Inf, mymean, mysd)      #  Should be 1
penorm(mean(yy), mymean, mysd)  #  Should be 0.5
abs(qenorm(0.5, mymean, mysd) - mean(yy))  #  Should be 0
abs(penorm(myexp, mymean, mysd) - my.p)    #  Should be 0
integrate(f = denorm, lower = -Inf, upper = Inf,
          mymean, mysd)  #  Should be 1
```

```
## Not run:
par(mfrow = c(2, 1))
yy <- seq(-3, 3, len = nn)
plot(yy, denorm(yy), type = "l", col="blue", xlab = "y", ylab = "g(y)",
     main = "g(y) for N(0,1); dotted green is f(y) = dnorm(y)")
lines(yy, dnorm(yy), col = "green", lty = "dotted", lwd = 2)  # 'original'

plot(yy, penorm(yy), type = "l", col = "blue", ylim = 0:1,
     xlab = "y", ylab = "G(y)", main = "G(y) for N(0,1)")
abline(v = 0, h = 0.5, col = "red", lty = "dashed")
lines(yy, pnorm(yy), col = "green", lty = "dotted", lwd = 2)
## End(Not run)
```

`Expectiles-sc.t2` *Expectiles/Quantiles of the Scaled Student t Distribution with 2 Df*

Description

Density function, distribution function, and quantile/expectile function and random generation for the scaled Student t distribution with 2 degrees of freedom.

Usage

```
dsc.t2(x, location = 0, scale = 1, log = FALSE)
psc.t2(q, location = 0, scale = 1, lower.tail = TRUE, log.p = FALSE)
qsc.t2(p, location = 0, scale = 1, lower.tail = TRUE, log.p = FALSE)
rsc.t2(n, location = 0, scale = 1)
```

Arguments

`x, q` Vector of expectiles/quantiles. See the terminology note below.

`p` Vector of probabilities. These should lie in $(0, 1)$.

`n, log` See `runif`.

`location, scale`
Location and scale parameters. The latter should have positive values. Values of these vectors are recyled.

`lower.tail, log.p`
Same meaning as in `pt` or `qt`.

Details

A Student-t distribution with 2 degrees of freedom and a scale parameter of `sqrt(2)` is equivalent to the standard form of this distribution (called Koenker's distribution below). Further details about this distribution are given in `sc.studentt2`.

Value

`dsc.t2(x)` gives the density function. `psc.t2(q)` gives the distribution function. `qsc.t2(p)` gives the expectile and quantile function. `rsc.t2(n)` gives n random variates.

Author(s)

T. W. Yee and Kai Huang

See Also

`dt`, `sc.studentt2`.

Examples

```
my.p <- 0.25; y <- rsc.t2(nn <- 5000)
(myexp <- qsc.t2(my.p))
sum(myexp - y[y <= myexp]) / sum(abs(myexp - y))  # Should be my.p
# Equivalently:
I1 <- mean(y <= myexp) * mean( myexp - y[y <= myexp])
I2 <- mean(y >  myexp) * mean(-myexp + y[y >  myexp])
I1 / (I1 + I2)  # Should be my.p
# Or:
I1 <- sum( myexp - y[y <= myexp])
I2 <- sum(-myexp + y[y >  myexp])

# Non-standard Koenker distribution
myloc <- 1; myscale <- 2
yy <- rsc.t2(nn, myloc, myscale)
(myexp <- qsc.t2(my.p, myloc, myscale))
sum(myexp - yy[yy <= myexp]) / sum(abs(myexp - yy))  # Should be my.p
psc.t2(mean(yy), myloc, myscale)  # Should be 0.5
abs(qsc.t2(0.5, myloc, myscale) - mean(yy))  # Should be 0
abs(psc.t2(myexp, myloc, myscale) - my.p)  # Should be 0
integrate(f = dsc.t2, lower = -Inf, upper = Inf,
          locat = myloc, scale = myscale)  # Should be 1

y <- seq(-7, 7, len = 201)
max(abs(dsc.t2(y) - dt(y / sqrt(2), df = 2) / sqrt(2)))  # Should be 0
## Not run:  plot(y, dsc.t2(y), type = "l", col = "blue", las = 1,
     ylim = c(0, 0.4), main = "Blue = Koenker; orange = N(0, 1)")
lines(y, dnorm(y), type = "l", col = "orange")
abline(h = 0, v = 0, lty = 2)
## End(Not run)
```

`Expectiles-Uniform` *Expectiles of the Uniform Distribution*

Description

Density function, distribution function, and expectile function and random generation for the distribution associated with the expectiles of a uniform distribution.

Usage

```
deunif(x, min = 0, max = 1, log = FALSE)
peunif(q, min = 0, max = 1, lower.tail = TRUE, log.p = FALSE)
qeunif(p, min = 0, max = 1, Maxit.nr = 10, Tol.nr = 1.0e-6,
       lower.tail = TRUE, log.p = FALSE)
reunif(n, min = 0, max = 1)
```

Arguments

`x, q`	Vector of expectiles. See the terminology note below.
`p`	Vector of probabilities. These should lie in $(0, 1)$.
`n, min, max, log`	See `runif`.
`lower.tail, log.p`	Same meaning as in `punif` or `qunif`.
`Maxit.nr`	Numeric. Maximum number of Newton-Raphson iterations allowed. A warning is issued if convergence is not obtained for all `p` values.
`Tol.nr`	Numeric. Small positive value specifying the tolerance or precision to which the expectiles are computed.

Details

Jones (1994) elucidated on the property that the expectiles of a random variable X with distribution function $F(x)$ correspond to the quantiles of a distribution $G(x)$ where G is related by an explicit formula to F. In particular, let y be the p-expectile of F. Then y is the p-quantile of G where

$$p = G(y) = (P(y) - yF(y))/(2[P(y) - yF(y)] + y - \mu),$$

and μ is the mean of X. The derivative of G is

$$g(y) = (\mu F(y) - P(y))/(2[P(y) - yF(y)] + y - \mu)^2.$$

Here, $P(y)$ is the partial moment $\int_{-\infty}^{y} x f(x)\, dx$ and $0 < p < 1$. The 0.5-expectile is the mean μ and the 0.5-quantile is the median.

A note about the terminology used here. Recall in the *S* language there are the `dpqr`-type functions associated with a distribution, e.g., `dunif`, `punif`, `qunif`, `runif`, for the uniform distribution. Here, `unif` corresponds to F and `eunif` corresponds to G. The addition of "e" (for *expectile*) is for the 'other' distribution associated with the parent distribution. Thus `deunif` is for g, `peunif` is for G, `qeunif` is for the inverse of G, `reunif` generates random variates from g.

For `qeunif` the Newton-Raphson algorithm is used to solve for y satisfying $p = G(y)$. Numerical problems may occur when values of `p` are very close to 0 or 1.

Value

`deunif(x)` gives the density function $g(x)$. `peunif(q)` gives the distribution function $G(q)$. `qeunif(p)` gives the expectile function: the expectile y such that $G(y) = p$. `reunif(n)` gives n random variates from G.

Author(s)

T. W. Yee and Kai Huang

References

Jones, M. C. (1994). Expectiles and M-quantiles are quantiles. *Statistics and Probability Letters*, **20**, 149–153.

See Also

`deexp`, `denorm`, `dunif`, `dsc.t2`.

Examples

```
my.p <- 0.25; y <- runif(nn <- 1000)
(myexp <- qeunif(my.p))
sum(myexp - y[y <= myexp]) / sum(abs(myexp - y))  # Should be my.p
# Equivalently:
I1 <- mean(y <= myexp) * mean( myexp - y[y <= myexp])
I2 <- mean(y >  myexp) * mean(-myexp + y[y >  myexp])
I1 / (I1 + I2)  # Should be my.p
# Or:
I1 <- sum( myexp - y[y <= myexp])
I2 <- sum(-myexp + y[y >  myexp])

# Non-standard uniform
mymin <- 1; mymax <- 8
yy <- runif(nn, mymin, mymax)
(myexp <- qeunif(my.p, mymin, mymax))
sum(myexp - yy[yy <= myexp]) / sum(abs(myexp - yy))  # Should be my.p
peunif(mymin, mymin, mymax)     #  Should be 0
peunif(mymax, mymin, mymax)     #  Should be 1
peunif(mean(yy), mymin, mymax)  #  Should be 0.5
abs(qeunif(0.5, mymin, mymax) - mean(yy))  #  Should be 0
abs(qeunif(0.5, mymin, mymax) - (mymin+mymax)/2)  #  Should be 0
abs(peunif(myexp, mymin, mymax) - my.p)  #  Should be 0
integrate(f = deunif, lower = mymin - 3, upper = mymax + 3,
          min = mymin, max = mymax)  # Should be 1

## Not run:
par(mfrow = c(2,1))
yy <- seq(0.0, 1.0, len = nn)
plot(yy, deunif(yy), type = "l", col = "blue", ylim = c(0, 2),
     xlab = "y", ylab = "g(y)", main = "g(y) for Uniform(0,1)")
lines(yy, dunif(yy), col = "green", lty = "dotted", lwd = 2)  # 'original'
```

```
plot(yy, peunif(yy), type = "l", col = "blue", ylim = 0:1,
     xlab = "y", ylab = "G(y)", main = "G(y) for Uniform(0,1)")
abline(a = 0.0, b = 1.0, col = "green", lty = "dotted", lwd = 2)
abline(v = 0.5, h = 0.5, col = "red", lty = "dashed")
## End(Not run)
```

---

expexpff *Exponentiated Exponential Distribution*

Description

Estimates the two parameters of the exponentiated exponential distribution by maximum likelihood estimation.

Usage

```
expexpff(lrate = "loglink", lshape = "loglink",
         irate = NULL, ishape = 1.1, tolerance = 1.0e-6, zero = NULL)
```

Arguments

`lshape, lrate`	Parameter link functions for the α and λ parameters. See `Links` for more choices. The defaults ensure both parameters are positive.
`ishape`	Initial value for the α parameter. If convergence fails try setting a different value for this argument.
`irate`	Initial value for the λ parameter. By default, an initial value is chosen internally using `ishape`.
`tolerance`	Numeric. Small positive value for testing whether values are close enough to 1 and 2.
`zero`	An integer-valued vector specifying which linear/additive predictors are modelled as intercepts only. The default is none of them. If used, choose one value from the set {1,2}. See `CommonVGAMffArguments` for more information.

Details

The exponentiated exponential distribution is an alternative to the Weibull and the gamma distributions. The formula for the density is

$$f(y; \lambda, \alpha) = \alpha\lambda(1 - \exp(-\lambda y))^{\alpha-1} \exp(-\lambda y)$$

where $y > 0$, $\lambda > 0$ and $\alpha > 0$. The mean of Y is $(\psi(\alpha + 1) - \psi(1))/\lambda$ (returned as the fitted values) where ψ is the digamma function. The variance of Y is $(\psi'(1) - \psi'(\alpha + 1))/\lambda^2$ where ψ' is the trigamma function.

This distribution has been called the two-parameter generalized exponential distribution by Gupta and Kundu (2006). A special case of the exponentiated exponential distribution: $\alpha = 1$ is the exponential distribution.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm and vgam.

Warning

Practical experience shows that reasonably good initial values really helps. In particular, try setting different values for the ishape argument if numerical problems are encountered or failure to convergence occurs. Even if convergence occurs try perturbing the initial value to make sure the global solution is obtained and not a local solution. The algorithm may fail if the estimate of the shape parameter is too close to unity.

Note

Fisher scoring is used, however, convergence is usually very slow. This is a good sign that there is a bug, but I have yet to check that the expected information is correct. Also, I have yet to implement Type-I right censored data using the results of Gupta and Kundu (2006).

Another algorithm for fitting this model is implemented in expexpff1.

Author(s)

T. W. Yee

References

Gupta, R. D. and Kundu, D. (2001). Exponentiated exponential family: an alternative to gamma and Weibull distributions, *Biometrical Journal*, **43**, 117–130.

Gupta, R. D. and Kundu, D. (2006). On the comparison of Fisher information of the Weibull and GE distributions, *Journal of Statistical Planning and Inference*, **136**, 3130–3144.

See Also

expexpff1, gammaR, weibullR, CommonVGAMffArguments.

Examples

```
# A special case: exponential data
edata <- data.frame(y = rexp(n <- 1000))
fit <- vglm(y ~ 1, fam = expexpff, data = edata, trace = TRUE, maxit = 99)
coef(fit, matrix = TRUE)
Coef(fit)


# Ball bearings data (number of million revolutions before failure)
edata <- data.frame(bbearings = c(17.88, 28.92, 33.00, 41.52, 42.12, 45.60,
48.80, 51.84, 51.96, 54.12, 55.56, 67.80, 68.64, 68.64,
68.88, 84.12, 93.12, 98.64, 105.12, 105.84, 127.92,
128.04, 173.40))
fit <- vglm(bbearings ~ 1, fam = expexpff(irate = 0.05, ish = 5),
            trace = TRUE, maxit = 300, data = edata)
```

```
coef(fit, matrix = TRUE)
Coef(fit)    # Authors get c(rate=0.0314, shape=5.2589)
logLik(fit)  # Authors get -112.9763


# Failure times of the airconditioning system of an airplane
eedata <- data.frame(acplane = c(23, 261, 87, 7, 120, 14, 62, 47,
225, 71, 246, 21, 42, 20, 5, 12, 120, 11, 3, 14,
71, 11, 14, 11, 16, 90, 1, 16, 52, 95))
fit <- vglm(acplane ~ 1, fam = expexpff(ishape = 0.8, irate = 0.15),
            trace = TRUE, maxit = 99, data = eedata)
coef(fit, matrix = TRUE)
Coef(fit)    # Authors get c(rate=0.0145, shape=0.8130)
logLik(fit)  # Authors get log-lik -152.264
```

---

expexpff1 *Exponentiated Exponential Distribution*

Description

Estimates the two parameters of the exponentiated exponential distribution by maximizing a profile (concentrated) likelihood.

Usage

```
expexpff1(lrate = "loglink", irate = NULL, ishape = 1)
```

Arguments

`lrate`	Parameter link function for the (positive) λ parameter. See `Links` for more choices.
`irate`	Initial value for the λ parameter. By default, an initial value is chosen internally using `ishape`.
`ishape`	Initial value for the α parameter. If convergence fails try setting a different value for this argument.

Details

See `expexpff` for details about the exponentiated exponential distribution. This family function uses a different algorithm for fitting the model. Given λ, the MLE of α can easily be solved in terms of λ. This family function maximizes a profile (concentrated) likelihood with respect to λ. Newton-Raphson is used, which compares with Fisher scoring with `expexpff`.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Warning

The standard errors produced by a `summary` of the model may be wrong.

Note

This family function works only for intercept-only models, i.e., `y ~ 1` where `y` is the response.

The estimate of α is attached to the `misc` slot of the object, which is a list and contains the component `shape`.

As Newton-Raphson is used, the working weights are sometimes negative, and some adjustment is made to these to make them positive.

Like `expexpff`, good initial values are needed. Convergence may be slow.

Author(s)

T. W. Yee

References

Gupta, R. D. and Kundu, D. (2001). Exponentiated exponential family: an alternative to gamma and Weibull distributions, *Biometrical Journal*, **43**, 117–130.

See Also

`expexpff`, `CommonVGAMffArguments`.

Examples

```
# Ball bearings data (number of million revolutions before failure)
edata <- data.frame(bbearings = c(17.88, 28.92, 33.00, 41.52, 42.12, 45.60,
48.80, 51.84, 51.96, 54.12, 55.56, 67.80, 68.64, 68.64,
68.88, 84.12, 93.12, 98.64, 105.12, 105.84, 127.92,
128.04, 173.40))
fit <- vglm(bbearings ~ 1, expexpff1(ishape = 4), trace = TRUE,
            maxit = 250, checkwz = FALSE, data = edata)
coef(fit, matrix = TRUE)
Coef(fit)  # Authors get c(0.0314, 5.2589) with log-lik -112.9763
logLik(fit)
fit@misc$shape  # Estimate of shape


# Failure times of the airconditioning system of an airplane
eedata <- data.frame(acplane = c(23, 261, 87, 7, 120, 14, 62, 47,
225, 71, 246, 21, 42, 20, 5, 12, 120, 11, 3, 14,
71, 11, 14, 11, 16, 90, 1, 16, 52, 95))
fit <- vglm(acplane ~ 1, expexpff1(ishape = 0.8), trace = TRUE,
            maxit = 50, checkwz = FALSE, data = eedata)
coef(fit, matrix = TRUE)
Coef(fit)  # Authors get c(0.0145, 0.8130) with log-lik -152.264
logLik(fit)
fit@misc$shape  # Estimate of shape
```

expgeom *The Exponential Geometric Distribution*

Description

Density, distribution function, quantile function and random generation for the exponential geometric distribution.

Usage

```
dexpgeom(x, scale = 1, shape, log = FALSE)
pexpgeom(q, scale = 1, shape)
qexpgeom(p, scale = 1, shape)
rexpgeom(n, scale = 1, shape)
```

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
n	number of observations. If length(n) > 1 then the length is taken to be the number required.
scale, shape	positive scale and shape parameters.
log	Logical. If log = TRUE then the logarithm of the density is returned.

Details

See expgeometric, the **VGAM** family function for estimating the parameters, for the formula of the probability density function and other details.

Value

dexpgeom gives the density, pexpgeom gives the distribution function, qexpgeom gives the quantile function, and rexpgeom generates random deviates.

Note

We define scale as the reciprocal of the scale parameter used by Adamidis and Loukas (1998).

Author(s)

J. G. Lauder and T. W. Yee

See Also

expgeometric, exponential, geometric.

Examples

```
## Not run:
shape <- 0.5; scale <- 1; nn <- 501
x <- seq(-0.10, 3.0, len = nn)
plot(x, dexpgeom(x, scale, shape), type = "l", las = 1, ylim = c(0, 2),
     ylab = paste("[dp]expgeom(shape = ", shape, ", scale = ", scale, ")"),
     col = "blue", cex.main = 0.8,
     main = "Blue is density, red is cumulative distribution function",
     sub = "Purple lines are the 10,20,...,90 percentiles")
lines(x, pexpgeom(x, scale, shape), col = "red")
probs <- seq(0.1, 0.9, by = 0.1)
Q <- qexpgeom(probs, scale, shape)
lines(Q, dexpgeom(Q, scale, shape), col = "purple", lty = 3, type = "h")
lines(Q, pexpgeom(Q, scale, shape), col = "purple", lty = 3, type = "h")
abline(h = probs, col = "purple", lty = 3)
max(abs(pexpgeom(Q, scale, shape) - probs))  # Should be 0

## End(Not run)
```

`expgeometric` *Exponential Geometric Distribution Family Function*

Description

Estimates the two parameters of the exponential geometric distribution by maximum likelihood estimation.

Usage

```
expgeometric(lscale = "loglink", lshape = "logitlink",
             iscale = NULL,   ishape = NULL,
             tol12 = 1e-05, zero = 1, nsimEIM = 400)
```

Arguments

`lscale, lshape`	Link function for the two parameters. See `Links` for more choices.
`iscale, ishape`	Numeric. Optional initial values for the scale and shape parameters.
`tol12`	Numeric. Tolerance for testing whether a parameter has value 1 or 2.
`zero, nsimEIM`	See `CommonVGAMffArguments`.

Details

The exponential geometric distribution has density function

$$f(y; c = scale, s = shape) = (1/c)(1-s)e^{-y/c}(1-se^{-y/c})^{-2}$$

where $y > 0$, $c > 0$ and $s \in (0,1)$. The mean, $(c(s-1)/s)\log(1-s)$ is returned as the fitted values. Note the median is $c\log(2-s)$. Simulated Fisher scoring is implemented.

Value

An object of class "vglmff" (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Note

We define `scale` as the reciprocal of the scale parameter used by Adamidis and Loukas (1998).

Author(s)

J. G. Lauder and T. W. Yee

References

Adamidis, K., Loukas, S. (1998). A lifetime distribution with decreasing failure rate. *Statistics and Probability Letters*, **39**, 35–42.

See Also

`dexpgeom`, `exponential`, `geometric`.

Examples

```
## Not run:
Scale <- exp(2); shape = logitlink(-1, inverse = TRUE);
edata <- data.frame(y = rexpgeom(n = 2000, scale = Scale, shape = shape))
fit <- vglm(y ~ 1, expgeometric, edata, trace = TRUE)
c(with(edata, mean(y)), head(fitted(fit), 1))
coef(fit, matrix = TRUE)
Coef(fit)
summary(fit)

## End(Not run)
```

`expint` *The Exponential Integral and Variants*

Description

Computes the exponential integral $Ei(x)$ for real values, as well as $\exp(-x) \times Ei(x)$ and $E_1(x)$ and their derivatives (up to the 3rd derivative).

Usage

```
expint(x, deriv = 0)
expexpint(x, deriv = 0)
expint.E1(x, deriv = 0)
```

Arguments

`x`	Numeric. Ideally a vector of positive reals.
`deriv`	Integer. Either 0, 1, 2 or 3.

Details

The exponential integral $Ei(x)$ function is the integral of $\exp(t)/t$ from 0 to x, for positive real x. The function $E_1(x)$ is the integral of $\exp(-t)/t$ from x to infinity, for positive real x.

Value

Function `expint(x, deriv = n)` returns the nth derivative of $Ei(x)$ (up to the 3rd), function `expexpint(x, deriv = n)` returns the nth derivative of $\exp(-x) \times Ei(x)$ (up to the 3rd), function `expint.E1(x, deriv = n)` returns the nth derivative of $E_1(x)$ (up to the 3rd).

Warning

These functions have not been tested thoroughly.

Author(s)

T. W. Yee has simply written a small wrapper function to call the NETLIB FORTRAN code. Xiangjie Xue modified the functions to calculate derivatives. Higher derivatives can actually be calculated—please let me know if you need it.

References

https://netlib.org/specfun/ei.

See Also

`log`, `exp`. There is also a package called **expint**.

Examples

```
 ## Not run:
par(mfrow = c(2, 2))
curve(expint, 0.01, 2, xlim = c(0, 2), ylim = c(-3, 5),
      las = 1, col = "orange")
abline(v = (-3):5, h = (-4):5, lwd = 2, lty = "dotted", col = "gray")
abline(h = 0, v = 0, lty = "dashed", col = "blue")

curve(expexpint, 0.01, 2, xlim = c(0, 2), ylim = c(-3, 2),
      las = 1, col = "orange")
abline(v = (-3):2, h = (-4):5, lwd = 2, lty = "dotted", col = "gray")
abline(h = 0, v = 0, lty = "dashed", col = "blue")

curve(expint.E1, 0.01, 2, xlim = c(0, 2), ylim = c(0, 5),
      las = 1, col = "orange")
abline(v = (-3):2, h = (-4):5, lwd = 2, lty = "dotted", col = "gray")
```

```
abline(h = 0, v = 0, lty = "dashed", col = "blue")

## End(Not run)
```

---

explink *Exponential Link Function*

Description

Computes the exponential transformation, including its inverse and the first two derivatives.

Usage

```
explink(theta, bvalue = NULL, inverse = FALSE, deriv = 0,
        short = TRUE, tag = FALSE)
```

Arguments

theta	Numeric or character. See below for further details.
bvalue	See cloglogink.
inverse, deriv, short, tag	Details at Links.

Details

The exponential link function is potentially suitable for parameters that are positive. Numerical values of `theta` close to negative or positive infinity may result in `0`, `Inf`, `-Inf`, `NA` or `NaN`.

Value

For `explink` with `deriv = 0`, the exponential of `theta`, i.e., `exp(theta)` when `inverse = FALSE`. And if `inverse = TRUE` then `log(theta)`; if `theta` is not positive then it will return `NaN`.

For `deriv = 1`, then the function returns *d* `eta` / *d* `theta` as a function of `theta` if `inverse = FALSE`, else if `inverse = TRUE` then it returns the reciprocal.

Here, all logarithms are natural logarithms, i.e., to base *e*.

Note

This function has particular use for computing quasi-variances when used with `rcim` and `uninormal`.

Numerical instability may occur when `theta` is close to negative or positive infinity. One way of overcoming this (one day) is to use `bvalue`.

Author(s)

Thomas W. Yee

See Also

Links, loglink, rcim, Qvar, uninormal.

Examples

```
theta <- rnorm(30)
explink(theta)
max(abs(explink(explink(theta), inverse = TRUE) - theta))  # 0?
```

---

explog — The Exponential Logarithmic Distribution

Description

Density, distribution function, quantile function and random generation for the exponential logarithmic distribution.

Usage

```
dexplog(x, scale = 1, shape, log = FALSE)
pexplog(q, scale = 1, shape)
qexplog(p, scale = 1, shape)
rexplog(n, scale = 1, shape)
```

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
n	number of observations. If length(n) > 1 then the length is taken to be the number required.
scale, shape	positive scale and shape parameters.
log	Logical. If log = TRUE then the logarithm of the density is returned.

Details

See explogff, the **VGAM** family function for estimating the parameters, for the formula of the probability density function and other details.

Value

dexplog gives the density, pexplog gives the distribution function, qexplog gives the quantile function, and rexplog generates random deviates.

Note

We define scale as the reciprocal of the scale parameter used by Tahmasabi and Rezaei (2008).

Author(s)

J. G. Lauder and T. W. Yee

See Also

explogff, exponential.

Examples

```
## Not run:
shape <- 0.5; scale <- 2; nn <- 501
x <- seq(-0.50, 6.0, len = nn)
plot(x, dexplog(x, scale, shape), type = "l", las = 1, ylim = c(0, 1.1),
     ylab = paste("[dp]explog(shape = ", shape, ", scale = ", scale, ")"),
     col = "blue", cex.main = 0.8,
     main = "Blue is density, orange is cumulative distribution function",
     sub = "Purple lines are the 10,20,...,90 percentiles")
lines(x, pexplog(x, scale, shape), col = "orange")
probs <- seq(0.1, 0.9, by = 0.1)
Q <- qexplog(probs, scale, shape = shape)
lines(Q, dexplog(Q, scale, shape = shape), col = "purple", lty = 3, type = "h")
lines(Q, pexplog(Q, scale, shape = shape), col = "purple", lty = 3, type = "h")
abline(h = probs, col = "purple", lty = 3)
max(abs(pexplog(Q, scale, shape = shape) - probs)) # Should be 0

## End(Not run)
```

---

explogff *Exponential Logarithmic Distribution Family Function*

Description

Estimates the two parameters of the exponential logarithmic distribution by maximum likelihood estimation.

Usage

```
explogff(lscale = "loglink", lshape = "logitlink",
         iscale = NULL,   ishape = NULL,
         tol12 = 1e-05, zero = 1, nsimEIM = 400)
```

Arguments

lscale, lshape	See CommonVGAMffArguments for information.
tol12	Numeric. Tolerance for testing whether a parameter has value 1 or 2.
iscale, ishape, zero, nsimEIM	See CommonVGAMffArguments.

Details

The exponential logarithmic distribution has density function

$$f(y; c, s) = (1/(-\log p))(((1/c)(1-s)e^{-y/c})/(1-(1-s)e^{-y/c}))$$

where $y > 0$, scale parameter $c > 0$, and shape parameter $s \in (0, 1)$. The mean, $(-polylog(2, 1-p)c)/\log(s)$ is *not* returned as the fitted values. Note the median is $c\log(1+\sqrt{s})$ and it is *currently* returned as the fitted values. Simulated Fisher scoring is implemented.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm and vgam.

Note

We define scale as the reciprocal of the rate parameter used by Tahmasabi and Sadegh (2008).

Yet to do: find a polylog() function.

Author(s)

J. G. Lauder and T. W .Yee

References

Tahmasabi, R., Sadegh, R. (2008). A two-parameter lifetime distribution with decreasing failure rate. *Computational Statistics and Data Analysis*, **52**, 3889–3901.

See Also

dexplog, exponential,

Examples

```
## Not run:  Scale <- exp(2); shape <- logitlink(-1, inverse = TRUE)
edata <- data.frame(y = rexplog(n = 2000, scale = Scale, shape = shape))
fit <- vglm(y ~ 1, explogff, data = edata, trace = TRUE)
c(with(edata, median(y)), head(fitted(fit), 1))
coef(fit, matrix = TRUE)
Coef(fit)
summary(fit)

## End(Not run)
```

`exponential` *Exponential Distribution*

Description

Maximum likelihood estimation for the exponential distribution.

Usage

```
exponential(link = "loglink", location = 0, expected = TRUE,
            type.fitted = c("mean", "percentiles", "Qlink"),
            percentiles = 50,
            ishrinkage = 0.95, parallel = FALSE, zero = NULL)
```

Arguments

`link` Parameter link function applied to the positive parameter $rate$. See `Links` for more choices.

`location` Numeric of length 1, the known location parameter, A, say.

`expected` Logical. If `TRUE` Fisher scoring is used, otherwise Newton-Raphson. The latter is usually faster.

`ishrinkage, parallel, zero`
See `CommonVGAMffArguments` for information.

`type.fitted, percentiles`
See `CommonVGAMffArguments` for information.

Details

The family function assumes the response Y has density

$$f(y) = \lambda \exp(-\lambda(y - A))$$

for $y > A$, where A is the known location parameter. By default, $A = 0$. Then $E(Y) = A + 1/\lambda$ and $Var(Y) = 1/\lambda^2$.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Note

Suppose $A = 0$. For a fixed time interval, the number of events is Poisson with mean λ if the time between events has a geometric distribution with mean λ^{-1}. The argument `rate` in `exponential` is the same as `rexp` etc. The argument `lambda` in `rpois` is somewhat the same as `rate` here.

Author(s)

T. W. Yee

References

Forbes, C., Evans, M., Hastings, N. and Peacock, B. (2011). *Statistical Distributions*, Hoboken, NJ, USA: John Wiley and Sons, Fourth edition.

See Also

amlexponential, gpd, laplace, expgeometric, explogff, poissonff, mix2exp, freund61, simulate.vlm, Exponential.

Examples

```
edata <- data.frame(x2 = runif(nn <- 100) - 0.5)
edata <- transform(edata, x3 = runif(nn) - 0.5)
edata <- transform(edata, eta = 0.2 - 0.7 * x2 + 1.9 * x3)
edata <- transform(edata, rate = exp(eta))
edata <- transform(edata, y = rexp(nn, rate = rate))
with(edata, stem(y))

fit.slow <- vglm(y ~ x2 + x3, exponential, data = edata, trace = TRUE)
fit.fast <- vglm(y ~ x2 + x3, exponential(exp = FALSE), data = edata,
                 trace = TRUE, crit = "coef")
coef(fit.slow, mat = TRUE)
summary(fit.slow)


# Compare results with a GPD. Has a threshold.
threshold <- 0.5
gdata <- data.frame(y1 = threshold + rexp(n = 3000, rate = exp(1.5)))

fit.exp <- vglm(y1 ~ 1, exponential(location = threshold), data = gdata)
coef(fit.exp, matrix = TRUE)
Coef(fit.exp)
logLik(fit.exp)

fit.gpd <- vglm(y1 ~ 1, gpd(threshold =  threshold), data = gdata)
coef(fit.gpd, matrix = TRUE)
Coef(fit.gpd)
logLik(fit.gpd)
```

---

exppois *The Exponential Poisson Distribution*

Description

Density, distribution function, quantile function and random generation for the exponential poisson distribution.

Usage

```
dexppois(x, rate = 1, shape, log = FALSE)
pexppois(q, rate = 1, shape, lower.tail = TRUE, log.p = FALSE)
qexppois(p, rate = 1, shape, lower.tail = TRUE, log.p = FALSE)
rexppois(n, rate = 1, shape)
```

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
n	number of observations. If length(n) > 1 then the length is taken to be the number required.
shape, rate	positive parameters.
log	Logical. If log = TRUE then the logarithm of the density is returned.
lower.tail, log.p	Same meaning as in pnorm or qnorm.

Details

See exppoisson, the **VGAM** family function for estimating the parameters, for the formula of the probability density function and other details.

Value

dexppois gives the density, pexppois gives the distribution function, qexppois gives the quantile function, and rexppois generates random deviates.

Author(s)

Kai Huang and J. G. Lauder

See Also

exppoisson.

Examples

```
## Not run:  rate <- 2; shape <- 0.5; nn <- 201
x <- seq(-0.05, 1.05, len = nn)
plot(x, dexppois(x, rate = rate, shape), type = "l", las = 1, ylim = c(0, 3),
     ylab = paste("fexppoisson(rate = ", rate, ", shape = ", shape, ")"),
     col = "blue", cex.main = 0.8,
     main = "Blue is the density, orange the cumulative distribution function",
     sub = "Purple lines are the 10,20,...,90 percentiles")
lines(x, pexppois(x, rate = rate, shape), col = "orange")
probs <- seq(0.1, 0.9, by = 0.1)
Q <- qexppois(probs, rate = rate, shape)
lines(Q, dexppois(Q, rate = rate, shape), col = "purple", lty = 3, type = "h")
lines(Q, pexppois(Q, rate = rate, shape), col = "purple", lty = 3, type = "h")
```

```
abline(h = probs, col = "purple", lty = 3); abline(h = 0, col = "gray50")
max(abs(pexppois(Q, rate = rate, shape) - probs))  # Should be 0

## End(Not run)
```

exppoisson *Exponential Poisson Distribution Family Function*

Description

Estimates the two parameters of the exponential Poisson distribution by maximum likelihood estimation.

Usage

```
exppoisson(lrate = "loglink", lshape = "loglink",
           irate = 2, ishape = 1.1,  zero = NULL)
```

Arguments

lshape, lrate	Link function for the two positive parameters. See Links for more choices.
ishape, irate	Numeric. Initial values for the shape and rate parameters. Currently this function is not intelligent enough to obtain better initial values.
zero	See CommonVGAMffArguments.

Details

The exponential Poisson distribution has density function

$$f(y; \beta = rate, \lambda = shape) = \frac{\lambda\beta}{1 - e^{-\lambda}} \, e^{-\lambda - \beta y + \lambda \exp{(-\beta y)}}$$

where $y > 0$, and the parameters shape, λ, and rate, β, are positive. The distribution implies a population facing discrete hazard rates which are multiples of a base hazard. This **VGAM** family function requires the hypergeo package (to use their genhypergeo function). The median is returned as the fitted value.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm and vgam.

Warning

This **VGAM** family function does not work properly!

Author(s)

J. G. Lauder, jamesglauder@gmail.com

References

Kus, C., (2007). A new lifetime distribution. *Computational Statistics and Data Analysis*, **51**, 4497–4509.

See Also

dexppois, exponential, poisson.

Examples

```
## Not run:
shape <- exp(1); rate <- exp(2)
rdata <- data.frame(y = rexppois(n = 1000, rate = rate, shape = shape))
library("hypergeo")  # Required!
fit <- vglm(y ~ 1, exppoisson, data = rdata, trace = FALSE, maxit = 1200)
c(with(rdata, median(y)), head(fitted(fit), 1))
coef(fit, matrix = TRUE)
Coef(fit)
summary(fit)

## End(Not run)
```

Extbetabinom   *The Beta-Binomial Distribution*

Description

Density, distribution function, quantile function and random generation for the extended beta-binomial distribution.

Usage

```
dextbetabinom(x, size, prob, rho = 0,
     log = FALSE, forbycol = TRUE)
pextbetabinom(q, size, prob, rho = 0,
     lower.tail = TRUE, forbycol = TRUE)
qextbetabinom(p, size, prob, rho = 0,
     forbycol = TRUE)
rextbetabinom(n, size, prob, rho = 0)
```

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
size	number of trials.
n	number of observations. Same as runif.
prob	the probability of success μ. Must be in the unit closed interval $[0, 1]$.

rho	the correlation parameter ρ, which may be negative for underdispersion or else be in the interval $[0, 1)$ for overdispersion. The default value of 0 corresponds to the usual binomial distribution with probability prob.
log, lower.tail	Same meaning as runif.
forbycol	Logical. A for loop cycles over either the rows or columns and this argument determines which. The rows are 1:length(x) and the columns are 0:max(size). The best choice is data set dependent.

Details

The *extended* beta-binomial distribution allows for a slightly negative correlation parameter between binary responses within a cluster (e.g., a litter). An exchangeable error structure with correlation ρ is assumed.

Value

dextbetabinom gives the density, pextbetabinom gives the distribution function, qextbetabinom gives the quantile function and rextbetabinom generates random deviates.

Warning

Setting rho = 1 is not recommended as NaN is returned, however the code may be modified in the future to handle this special case.

Note

Currently most of the code is quite slow. Speed improvements are a future project. Use forbycol optimally.

See Also

extbetabinomial, Betabinom, Binomial.

Examples

```
set.seed(1); rextbetabinom(10, 100, 0.5)
set.seed(1);        rbinom(10, 100, 0.5)  # Same

## Not run: N <- 9; xx <- 0:N; prob <- 0.5; rho <- -0.02
dy <- dextbetabinom(xx, N, prob, rho)
barplot(rbind(dy, dbinom(xx, size = N, prob)),
  beside = TRUE, col = c("blue","green"), las = 1,
  main = paste0("Beta-binom(size=", N,
  ", prob=", prob, ", rho=", rho, ") (blue) vs\n",
  " Binom(size=", N, ", prob=", prob, ") (green)"),
  names.arg = as.character(xx), cex.main = 0.8)
sum(dy * xx)  # Check expected values are equal
sum(dbinom(xx, size = N, prob = prob) * xx)
cumsum(dy) - pextbetabinom(xx, N, prob, rho)  # 0?
```

```
## End(Not run)
```

---

extbetabinomial *Extended Beta-binomial Distribution Family Function*

Description

Fits an extended beta-binomial distribution by maximum likelihood estimation. The two parameters here are the mean and correlation coefficient.

Usage

```
extbetabinomial(lmu = "logitlink", lrho = "cloglink",
      zero = "rho", irho = 0, grho = c(0, 0.05, 0.1, 0.2),
      vfl = FALSE, Form2 = NULL,
      imethod = 1, ishrinkage = 0.95)
```

Arguments

lmu, lrho	Link functions applied to the two parameters. See Links for more choices. The first default ensure the mean remain in $(0, 1)$, while the second allows for a slightly negative correlation parameter: you could say it lies in $(\max(-\mu/(N - \mu - 1), -(1 - \mu)/(N - (1 - \mu) - 1)), 1)$ where μ is the mean (probability) and N is size. See below for details. For lrho, cloglink is a good choice because it handles parameter values from 1 downwards. Other good choices include logofflink(offset = 1) and rhobitlink.
irho, grho	The first is similar to betabinomial and it is a good idea to use this argument because to conduct a grid search based on grho is expensive. The default is effectively a binomial distribution. Set irho = NULL to perform a grid search which is more reliable but slow.
imethod	Similar to betabinomial.
zero	Similar to betabinomial. Also, see CommonVGAMffArguments for more information. Modelling rho with covariates requires large samples.
ishrinkage	See betabinomial and CommonVGAMffArguments for information.
vfl, Form2	See CommonVGAMffArguments. If vfl = TRUE then Form2 should be a formula specifying the terms for η_2 and all others are used for μ. It is similar to uninormal. If these arguments are used then cbind(0, log(size1 / (size1 - 1))) should be used as an offset, and set zero = NULL too.

Details

The *extended* beta-binomial distribution (EBBD) proposed by Prentice (1986) allows for a slightly negative correlation parameter whereas the ordinary BBD betabinomial only allows values in $(0, 1)$ so it handles overdispersion only. When negative, the data is underdispersed relative to an ordinary binomial distribution.

Argument `rho` is used here for the δ used in Prentice (1986) because it is the correlation between the (almost) Bernoulli trials. (They are actually simple binary variates.) We use here N for the number of trials (e.g., litter size), $T = NY$ is the number of successes, and p (or μ) is the probability of a success (e.g., a malformation). That is, Y is the *proportion* of successes. Like `binomialff`, the fitted values are the estimated probability of success (i.e., $E[Y]$ and not $E[T]$) and the prior weights N are attached separately on the object in a slot.

The probability function is difficult to write but it involves three series of products. Recall $Y = T/N$ is the real response being modelled, where T is the (total) sum of N correlated (almost) Bernoulli trials.

The default model is $\eta_1 = logit(\mu)$ and $\eta_2 = clog(\rho)$ because the first parameter lies between 0 and 1. The second link is `cloglink`. The mean of Y is $p = \mu$ and the variance of Y is $\mu(1-\mu)(1+(N-1)\rho)/N$. Here, the correlation ρ may be slightly negative and is the correlation between the N individuals within a litter. A *litter effect* is typically reflected by a positive value of ρ and corresponds to *overdispersion* with respect to the binomial distribution. Thus an *exchangeable* error structure is assumed between units within a litter for the EBBD.

This family function uses Fisher scoring. Elements of the second-order expected derivatives are computed numerically, which may fail for models very near the boundary of the parameter space. Usually, the computations are expensive for large N because of a `for` loop, so it may take a long time.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`.

Suppose `fit` is a fitted EBB model. Then `depvar(fit)` are the sample proportions y, `fitted(fit)` returns estimates of $E(Y)$, and `weights(fit, type = "prior")` returns the number of trials N.

Warning

Modelling `rho` using covariates well requires much data so it is usually best to leave `zero` alone. It is good to set `trace = TRUE` and play around with `irho` if there are problems achieving convergence. Convergence problems will occur when the estimated `rho` is close to the lower bound, i.e., the underdispersion is almost too severe for the EBB to cope.

Note

This function is recommended over `betabinomial` and `betabinomialff`. It processes the input in the same way as `binomialff`. But it does not handle the case $N \leq 2$ very well because there are two parameters to estimate, not one, for each row of the input. Cases where $N > 2$ can be selected via the `subset` argument of `vglm`.

Author(s)

T. W. Yee

References

Prentice, R. L. (1986). Binary regression using an extended beta-binomial distribution, with discussion of correlation induced by covariate measurement errors. *Journal of the American Statistical Association*, **81**, 321–327.

See Also

Extbetabinom, betabinomial, betabinomialff, binomialff, dirmultinomial, cloglink, lirat.

Examples

```
# Example 1
edata <- data.frame(N = 10, mu = 0.5, rho = 0.1)
edata <- transform(edata,
      y = rextbetabinom(100, N, mu, rho = rho))
fit1 <- vglm(cbind(y, N-y) ~ 1, extbetabinomial, edata, trace = TRUE)
coef(fit1, matrix = TRUE)
Coef(fit1)
head(cbind(depvar(fit1), weights(fit1, type = "prior")))

# Example 2: VFL model
## Not run: N <- size1 <- 10; nn <- 2000; set.seed(1)
edata <-  # Generate the data set. Expensive.
    data.frame(x2 = runif(nn),
               ooo =  log(size1 / (size1 - 1)))
edata <- transform(edata, x1copy = 1, x2copy = x2,
  y2 = rextbetabinom(nn, size1,  # Expensive
         logitlink(1 + x2, inverse = TRUE),
         cloglink(ooo + 1 - 0.5 * x2, inv = TRUE)))
fit2 <- vglm(data = edata,
        cbind(y2, N - y2) ~ x2 + x1copy + x2copy,
        extbetabinomial(zero = NULL, vfl = TRUE,
                 Form2 = ~ x1copy + x2copy - 1),
        offset = cbind(0, ooo), trace = TRUE)
coef(fit2, matrix = TRUE)
wald.stat(fit2, values0 = c(1, 1, -0.5))

## End(Not run)
```

---

extlogF1 *Extended log-F Distribution Family Function*

Description

Maximum likelihood estimation of the 1-parameter extended log-F distribution.

Usage

```
extlogF1(tau = c(0.25, 0.5, 0.75), parallel = TRUE ~ 0,
         seppar = 0, tol0 = -0.001,
         llocation = "identitylink", ilocation = NULL,
         lambda.arg = NULL, scale.arg = 1, ishrinkage = 0.95,
         digt = 4, idf.mu = 3, imethod = 1)
```

Arguments

`tau` Numeric, the desired quantiles. A strictly increasing sequence, each value must be in $(0, 1)$. The default values are the three quartiles, matching `lms.bcn`.

`parallel` Similar to `alaplace1`, applying to the location parameters. One can try fix up the quantile-crossing problem after fitting the model by calling `fix.crossing`. Use `is.crossing` to see if there is a problem. The default for `parallel` is totally `FALSE`, i.e., `FALSE` for every variable including the intercept. Quantile-crossing can occur when values of `tau` are too close, given the data. How the quantiles are modelled with respect to the covariates also has a big effect, e.g., if they are too flexible or too inflexible then the problem is likely to occur. For example, using `bs` with `df = 10` is likely to create problems.

Setting `parallel = TRUE` results in a totally parallel model; *all* quantiles are parallel and this assumption can be too strong for some data sets. Instead, `fix.crossing` only repairs the quantiles that cross. So one must carefully choose values of `tau` for fitting the original fit.

`seppar, tol0` Numeric, both of unit length and nonnegative, the separation and shift parameters. If `seppar` is positive then any crossing quantile is penalized by the difference cubed multiplied by `seppar`. The log-likelihood subtracts the penalty. The shift parameter ensures that the result is strictly noncrossing when `seppar` is large enough; otherwise if `tol0 = 0` and `seppar` is large then the crossing quantiles remain crossed even though the offending amount becomes small but never exactly 0. Informally, `tol0` pushes the adjustment enough so that `is.crossing` should return `FALSE`.

If `tol0` is positive then that is the shift in absolute terms. But `tol0` may be assigned a negative value, in which case it is interpreted multiplicatively *relative* to the midspread of the response; `tol0 <- abs(tol0) * midspread`. Regardless, `fit@extra$tol0` is the amount in absolute terms.

If avoiding the quantile crossing problem is of concern to you, try increasing `seppar` to decrease the amount of crossing. Probably it is best to choose the smallest value of `seppar` so that `is.crossing` returns `FALSE`. Increasing `tol0` relatively or absolutely means the fitted quantiles are allowed to move apart more. However, `tau` must be considered when choosing `tol0`.

`llocation, ilocation`
See `Links` for more choices and `CommonVGAMffArguments` for more information. Choosing `loglink` should usually be good for counts. And choosing `logitlink` should be a reasonable for proportions. However, avoid choosing `tau` values close to the boundary, for example, if p_0 is the proportion of 0s then choose $p_0 \ll \tau$. For proportions grouped data is much better than ungrouped

data, and the bigger the groups the more the granularity so that the empirical proportion can approximate `tau` more closely.

`lambda.arg`
: Positive tuning parameter which controls the sharpness of the cusp. The limit as it approaches 0 is probably very similar to `dalap`. The default is to choose the value internally. If `scale.arg` increases, then probably `lambda.arg` needs to increase accordingly. If `lambda.arg` is too large then the empirical quantiles may not be very close to `tau`. If `lambda.arg` is too close to 0 then the convergence behaviour will not be good and local solutions found, as well as numerical problems in general. Monitoring convergence is recommended when varying `lambda.arg`.

`scale.arg`
: Positive scale parameter and sometimes called `scale`. The transformation used is `(y - location) / scale`. This function should be okay for response variables having a moderate range (0–100, say), but if very different from this then experimenting with this argument will be a good idea.

`ishrinkage, idf.mu, digt`
: Similar to `alaplace1`.

`imethod`
: Initialization method. Either the value 1, 2, or .... See `CommonVGAMffArguments` for more information.

Details

This is an experimental family function for quantile regression. Fasiolo et al. (2020) propose an *extended* log-F distribution (ELF) however this family function only estimates the location parameter. The distribution has a scale parameter which can be inputted (default value is unity). One location parameter is estimated for each `tau` value and these are the estimated quantiles. For quantile regression it is not necessary to estimate the scale parameter since the log-likelihood function is triangle shaped.

The ELF is used as an approximation of the asymmetric Laplace distribution (ALD). The latter cannot be estimated properly using Fisher scoring/IRLS but the ELF holds promise because it has continuous derivatives and therefore fewer problems with the regularity conditions. Because the ELF is fitted to data to obtain an empirical result the convergence behaviour may not be gentle and smooth. Hence there is a function-specific control function called `extlogF1.control` which has something like `stepsize = 0.5` and `maxits = 100`. It has been found that slowing down the rate of convergence produces greater stability during the estimation process. Regardless, convergence should be monitored carefully always.

This function accepts a vector response but not a matrix response.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Note

Changes will occur in the future to fine-tune things. In general setting `trace = TRUE` is strongly encouraged because it is needful to check that convergence occurs properly.

If `seppar > 0` then `logLik(fit)` will return the penalized log-likelihood.

Author(s)

Thomas W. Yee

References

Fasiolo, M., Wood, S. N., Zaffran, M., Nedellec, R. and Goude, Y. (2020). Fast calibrated additive quantile regression. *J. Amer. Statist. Assoc.*, in press.

Yee, T. W. (2020). On quantile regression based on the 1-parameter extended log-F distribution. *In preparation*.

See Also

dextlogF, is.crossing, fix.crossing, eCDF, vglm.control, logF, alaplace1, dalap, lms.bcn.

Examples

```
nn <- 1000; mytau <- c(0.25, 0.75)
edata <- data.frame(x2 = sort(rnorm(nn)))
edata <- transform(edata, y1 = 1 + x2  + rnorm(nn, sd = exp(-1)),
          y2 = cos(x2) / (1 + abs(x2)) + rnorm(nn, sd = exp(-1)))
fit1 <- vglm(y1 ~ x2, extlogF1(tau = mytau), data = edata)  # trace = TRUE
fit2 <- vglm(y2 ~ bs(x2, 6), extlogF1(tau = mytau), data = edata)
coef(fit1, matrix = TRUE)
fit2@extra$percentile  # Empirical percentiles here
summary(fit2)
c(is.crossing(fit1), is.crossing(fit2))
head(fitted(fit1))
## Not run: plot(y2 ~ x2, edata, col = "blue")
matlines(with(edata, x2), fitted(fit2), col="orange", lty = 1, lwd = 2)
## End(Not run)
```

---

familyname   *Family Function Name*

Description

Extractor function for the name of the family function of an object in the **VGAM** package.

Usage

```
familyname(object, ...)
familyname.vlm(object, all = FALSE, ...)
```

Arguments

`object`	Some **VGAM** object, for example, having class vglmff-class.
`all`	If `all = TRUE` then all of the `vfamily` slot is returned; this contains subclasses the object might have. The default is the return the first value only.
`...`	Other possible arguments for the future.

Details

Currently **VGAM** implements over 150 family functions. This function returns the name of the function assigned to the `family` argument, for modelling functions such as `vglm` and `vgam`. Sometimes a slightly different answer is returned, e.g., `propodds` really calls `cumulative` with some arguments set, hence the output returned by this function is `"cumulative"` (note that one day this might change, however).

Value

A character string or vector.

Note

Arguments used in the invocation are not included. Possibly this is something to be done in the future.

See Also

`vglmff-class`, `vglm-class`.

Examples

```
pneumo <- transform(pneumo, let = log(exposure.time))
fit1 <- vglm(cbind(normal, mild, severe) ~ let,
             cumulative(parallel = TRUE, reverse = TRUE), data = pneumo)
familyname(fit1)
familyname(fit1, all = TRUE)
familyname(propodds())  # "cumulative"
```

`Felix` *The Felix Distribution*

Description

Density for the Felix distribution.

Usage

```
dfelix(x, rate = 0.25, log = FALSE)
```

Arguments

`x`	vector of quantiles.
`rate`	See `felix`.
`log`	Logical. If `log = TRUE` then the logarithm of the density is returned.

Details

See felix, the **VGAM** family function for estimating the parameter, for the formula of the probability density function and other details.

Value

dfelix gives the density.

Warning

The default value of rate is subjective.

Author(s)

T. W. Yee

See Also

felix.

Examples

```
## Not run:
rate <- 0.25; x <- 1:15
plot(x, dfelix(x, rate), type = "h", las = 1, col = "blue",
     ylab = paste("dfelix(rate=", rate, ")"),
     main = "Felix density function")

## End(Not run)
```

---

felix *Felix Distribution Family Function*

Description

Estimates the parameter of a Felix distribution by maximum likelihood estimation.

Usage

```
felix(lrate = extlogitlink(min = 0, max = 0.5), imethod = 1)
```

Arguments

lrate	Link function for the parameter, called a below; see Links for more choices and for general information.
imethod	See CommonVGAMffArguments. Valid values are 1, 2, 3 or 4.

Details

The Felix distribution is an important basic Lagrangian distribution. The density function is

$$f(y; a) = \frac{1}{((y-1)/2)!} y^{(y-3)/2} a^{(y-1)/2} \exp(-ay)$$

where $y = 1, 3, 5, \ldots$ and $0 < a < 0.5$. The mean is $1/(1 - 2a)$ (returned as the fitted values). Fisher scoring is implemented.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm and vgam.

Author(s)

T. W. Yee

References

Consul, P. C. and Famoye, F. (2006). *Lagrangian Probability Distributions*, Boston, USA: Birkhauser.

See Also

dfelix, borel.tanner.

Examples

```
fdata <- data.frame(y = 2 * rpois(n = 200, 1) + 1)  # Not real data!
fit <- vglm(y ~ 1, felix, data = fdata, trace = TRUE, crit = "coef")
coef(fit, matrix = TRUE)
Coef(fit)
summary(fit)
```

---

fff *F Distribution Family Function*

Description

Maximum likelihood estimation of the (2-parameter) F distribution.

Usage

```
fff(link = "loglink", idf1 = NULL, idf2 = NULL, nsimEIM = 100,
    imethod = 1, zero = NULL)
```

Arguments

`link`	Parameter link function for both parameters. See `Links` for more choices. The default keeps the parameters positive.
`idf1, idf2`	Numeric and positive. Initial value for the parameters. The default is to choose each value internally.
`nsimEIM, zero`	See `CommonVGAMffArguments` for more information.
`imethod`	Initialization method. Either the value 1 or 2. If both fail try setting values for `idf1` and `idf2`.

Details

The F distribution is named after Fisher and has a density function that has two parameters, called `df1` and `df2` here. This function treats these degrees of freedom as *positive reals* rather than integers. The mean of the distribution is $df2/(df2-2)$ provided $df2 > 2$, and its variance is $2df2^2(df1+df2-2)/(df1(df2-2)^2(df2-4))$ provided $df2 > 4$. The estimated mean is returned as the fitted values. Although the F distribution can be defined to accommodate a non-centrality parameter `ncp`, it is assumed zero here. Actually it shouldn't be too difficult to handle any known `ncp`; something to do in the short future.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Warning

Numerical problems will occur when the estimates of the parameters are too low or too high.

Author(s)

T. W. Yee

References

Forbes, C., Evans, M., Hastings, N. and Peacock, B. (2011). *Statistical Distributions*, Hoboken, NJ, USA: John Wiley and Sons, Fourth edition.

See Also

`FDist`.

Examples

```
## Not run:
fdata <- data.frame(x2 = runif(nn <- 2000))
fdata <- transform(fdata, df1 = exp(2+0.5*x2),
                          df2 = exp(2-0.5*x2))
fdata <- transform(fdata, y   = rf(nn, df1, df2))
fit <- vglm(y  ~ x2, fff, data = fdata, trace = TRUE)
```

```
coef(fit, matrix = TRUE)

## End(Not run)
```

---

fill1 *Creates a Matrix of Appropriate Dimension*

Description

A support function for the argument `xij`, it generates a matrix of an appropriate dimension.

Usage

```
fill1(x, values = 0, ncolx = ncol(x))
```

Arguments

`x`	A vector or matrix which is used to determine the dimension of the answer, in particular, the number of rows. After converting `x` to a matrix if necessary, the answer is a matrix of `values` values, of dimension `nrow(x)` by `ncolx`.
`values`	Numeric. The answer contains these values, which are recycled *columnwise* if necessary, i.e., as `matrix(values, ..., byrow=TRUE)`.
`ncolx`	The number of columns of the returned matrix. The default is the number of columns of `x`.

Details

The `xij` argument for `vglm` allows the user to input variables specific to each linear/additive predictor. For example, consider the bivariate logit model where the first/second linear/additive predictor is the logistic regression of the first/second binary response respectively. The third linear/additive predictor is `log(OR) = eta3`, where `OR` is the odds ratio. If one has ocular pressure as a covariate in this model then `xij` is required to handle the ocular pressure for each eye, since these will be different in general. [This contrasts with a variable such as `age`, the age of the person, which has a common value for both eyes.] In order to input these data into `vglm` one often finds that functions `fill1`, `fill2`, etc. are useful.

All terms in the `xij` and `formula` arguments in `vglm` must appear in the `form2` argument too.

Value

`matrix(values, nrow=nrow(x), ncol=ncolx)`, i.e., a matrix consisting of values `values`, with the number of rows matching `x`, and the default number of columns is the number of columns of `x`.

Note

The effect of the `xij` argument is after other arguments such as `exchangeable` and `zero`. Hence `xij` does not affect constraint matrices.

Additionally, there are currently 3 other identical `fill1` functions, called `fill2`, `fill3` and `fill4`; if you need more then assign `fill5 = fill6 = fill1` etc. The reason for this is that if more than one `fill1` function is needed then they must be unique. For example, if $M = 4$ then `xij = list(op ~ lop + rop + fill1(mop) + fill1(mop))` would reduce to `xij = list(op ~ lop + rop + fill1(mop))`, whereas `xij = list(op ~ lop + rop + fill1(mop) + fill2(mop))` would retain all M terms, which is needed.

In Examples 1 to 3 below, the `xij` argument illustrates covariates that are specific to a linear predictor. Here, `lop`/`rop` are the ocular pressures of the left/right eye in an artificial dataset, and `mop` is their mean. Variables `leye` and `reye` might be the presence/absence of a particular disease on the LHS/RHS eye respectively.

In Example 3, the `xij` argument illustrates fitting the (exchangeable) model where there is a common smooth function of the ocular pressure. One should use regression splines since `s` in `vgam` does not handle the `xij` argument. However, regression splines such as `bs` and `ns` need to have the same basis functions here for both functions, and Example 3 illustrates a trick involving a function `BS` to obtain this, e.g., same knots. Although regression splines create more than a single column per term in the model matrix, `fill1(BS(lop,rop))` creates the required (same) number of columns.

Author(s)

T. W. Yee

See Also

`vglm.control`, `vglm`, `multinomial`, `Select`.

Examples

```
fill1(runif(5))
fill1(runif(5), ncol = 3)
fill1(runif(5), val = 1, ncol = 3)

# Generate (independent) eyes data for the examples below; OR=1.
nn <- 1000  # Number of people
eyesdata <- data.frame(lop = round(runif(nn), 2),
                       rop = round(runif(nn), 2),
                       age = round(rnorm(nn, 40, 10)))
eyesdata <- transform(eyesdata,
  mop = (lop + rop) / 2,        # Mean ocular pressure
  op  = (lop + rop) / 2,        # Value unimportant unless plotting
# op  =  lop,                   # Choose this if plotting
  eta1 = 0 - 2*lop + 0.04*age,  # Linear predictor for left eye
  eta2 = 0 - 2*rop + 0.04*age)  # Linear predictor for right eye
eyesdata <- transform(eyesdata,
  leye = rbinom(nn, size=1, prob = logitlink(eta1, inverse = TRUE)),
  reye = rbinom(nn, size=1, prob = logitlink(eta2, inverse = TRUE)))
```

```
# Example 1. All effects are linear.
fit1 <- vglm(cbind(leye,reye) ~ op + age,
             family = binom2.or(exchangeable = TRUE, zero = 3),
             data = eyesdata, trace = TRUE,
             xij = list(op ~ lop + rop + fill1(lop)),
             form2 =  ~ op + lop + rop + fill1(lop) + age)
head(model.matrix(fit1, type = "lm"))   # LM model matrix
head(model.matrix(fit1, type = "vlm"))  # Big VLM model matrix
coef(fit1)
coef(fit1, matrix = TRUE)  # Unchanged with 'xij'
constraints(fit1)
max(abs(predict(fit1)-predict(fit1, new = eyesdata)))  # Okay
summary(fit1)
## Not run:
plotvgam(fit1,
     se = TRUE)  # Wrong, e.g., coz it plots against op, not lop.
# So set op = lop in the above for a correct plot.

## End(Not run)

# Example 2. This uses regression splines on ocular pressure.
# It uses a trick to ensure common basis functions.
BS <- function(x, ...)
  sm.bs(c(x,...), df = 3)[1:length(x), , drop = FALSE]  # trick

fit2 <-
  vglm(cbind(leye,reye) ~ BS(lop,rop) + age,
       family = binom2.or(exchangeable = TRUE, zero = 3),
       data = eyesdata, trace = TRUE,
       xij = list(BS(lop,rop) ~ BS(lop,rop) +
                                BS(rop,lop) +
                                fill1(BS(lop,rop))),
       form2 = ~  BS(lop,rop) + BS(rop,lop) + fill1(BS(lop,rop)) +
                        lop + rop + age)
head(model.matrix(fit2, type =  "lm"))  # LM model matrix
head(model.matrix(fit2, type = "vlm"))  # Big VLM model matrix
coef(fit2)
coef(fit2, matrix = TRUE)
summary(fit2)
fit2@smart.prediction
max(abs(predict(fit2) - predict(fit2, new = eyesdata)))  # Okay
predict(fit2, new = head(eyesdata))  # OR is 'scalar' as zero=3
max(abs(head(predict(fit2)) -
             predict(fit2, new = head(eyesdata))))  # Should be 0
## Not run:
plotvgam(fit2, se = TRUE, xlab = "lop")  # Correct

## End(Not run)

# Example 3. Capture-recapture model with ephemeral and enduring
# memory effects. Similar to Yang and Chao (2005), Biometrics.
deermice <- transform(deermice, Lag1 = y1)
M.tbh.lag1 <-
```

```
  vglm(cbind(y1, y2, y3, y4, y5, y6) ~ sex + weight + Lag1,
       posbernoulli.tb(parallel.t = FALSE ~ 0,
                       parallel.b = FALSE ~ 0,
                       drop.b = FALSE ~ 1),
       xij = list(Lag1 ~ fill1(y1) + fill1(y2) + fill1(y3) +
                         fill1(y4) + fill1(y5) + fill1(y6) +
                         y1 + y2 + y3 + y4 + y5),
       form2 = ~ sex + weight + Lag1 +
                 fill1(y1) + fill1(y2) + fill1(y3) + fill1(y4) +
                 fill1(y5) + fill1(y6) +
                 y1 + y2 + y3 + y4 + y5 + y6,
       data = deermice, trace = TRUE)
coef(M.tbh.lag1)
```

---

finney44 *Toxicity trial for insects*

Description

A data frame of a toxicity trial.

Usage

```
data(finney44)
```

Format

A data frame with 6 observations on the following 3 variables.

`pconc` a numeric vector, percent concentration of pyrethrins.

`hatched` number of eggs that hatched.

`unhatched` number of eggs that did not hatch.

Details

Finney (1944) describes a toxicity trial of five different concentrations of pyrethrins (percent) plus a control that were administered to eggs of *Ephestia kuhniella*. The natural mortality rate is large, and a common adjustment is to use Abbott's formula.

References

Finney, D. J. (1944). The application of the probit method to toxicity test data adjusted for mortality in the controls. *Annals of Applied Biology*, **31**, 68–74.

Abbott, W. S. (1925). A method of computing the effectiveness of an insecticide. *Journal of Economic Entomology*, 18, 265–7.

Examples

```
data(finney44)
transform(finney44, mortality = unhatched / (hatched + unhatched))
```

fisherzlink *Fisher's Z Link Function*

Description

Computes the Fisher Z transformation, including its inverse and the first two derivatives.

Usage

```
fisherzlink(theta, bminvalue = NULL, bmaxvalue = NULL,
            inverse = FALSE, deriv = 0, short = TRUE, tag = FALSE)
```

Arguments

`theta` Numeric or character. See below for further details.

`bminvalue, bmaxvalue`
Optional boundary values. Values of `theta` which are less than or equal to -1 can be replaced by `bminvalue` before computing the link function value. Values of `theta` which are greater than or equal to 1 can be replaced by `bmaxvalue` before computing the link function value. See `Links`.

`inverse, deriv, short, tag`
Details at `Links`.

Details

The `fisherz` link function is commonly used for parameters that lie between -1 and 1. Numerical values of `theta` close to -1 or 1 or out of range result in `Inf`, `-Inf`, `NA` or `NaN`.

Value

For `deriv = 0`, `0.5 * log((1+theta)/(1-theta))` (same as `atanh(theta)`) when `inverse = FALSE`, and if `inverse = TRUE` then `(exp(2*theta)-1)/(exp(2*theta)+1)` (same as `tanh(theta)`).

For `deriv = 1`, then the function returns *d* `eta` / *d* `theta` as a function of `theta` if `inverse = FALSE`, else if `inverse = TRUE` then it returns the reciprocal.

Here, all logarithms are natural logarithms, i.e., to base *e*.

Note

Numerical instability may occur when `theta` is close to -1 or 1. One way of overcoming this is to use, e.g., `bminvalue`.

The link function `rhobitlink` is very similar to `fisherzlink`, e.g., just twice the value of `fisherzlink`. This link function may be renamed to `atanhlink` in the near future.

Author(s)

Thomas W. Yee

References

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*, 2nd ed. London: Chapman & Hall.

See Also

`Links`, `rhobitlink`, `logitlink`.

Examples

```
theta <- seq(-0.99, 0.99, by = 0.01)
y <- fisherzlink(theta)
## Not run:  plot(theta, y, type = "l", las = 1, ylab = "",
   main = "fisherzlink(theta)", col = "blue")
abline(v = (-1):1, h = 0, lty = 2, col = "gray")
## End(Not run)

x <- c(seq(-1.02, -0.98, by = 0.01), seq(0.97, 1.02, by = 0.01))
fisherzlink(x)  # Has NAs
fisherzlink(x, bminvalue = -1 + .Machine$double.eps,
               bmaxvalue =  1 - .Machine$double.eps)  # Has no NAs
```

`Fisk` *The Fisk Distribution*

Description

Density, distribution function, quantile function and random generation for the Fisk distribution with shape parameter a and scale parameter `scale`.

Usage

```
dfisk(x, scale = 1, shape1.a, log = FALSE)
pfisk(q, scale = 1, shape1.a, lower.tail = TRUE, log.p = FALSE)
qfisk(p, scale = 1, shape1.a, lower.tail = TRUE, log.p = FALSE)
rfisk(n, scale = 1, shape1.a)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. If `length(n)` > 1 then the length is taken to be the number required.
`shape1.a`	shape parameter.
`scale`	scale parameter.
`log`	Logical. If `log` = `TRUE` then the logarithm of the density is returned.
`lower.tail, log.p`	Same meaning as in `pnorm` or `qnorm`.

Details

See fisk, which is the **VGAM** family function for estimating the parameters by maximum likelihood estimation.

Value

dfisk gives the density, pfisk gives the distribution function, qfisk gives the quantile function, and rfisk generates random deviates.

Note

The Fisk distribution is a special case of the 4-parameter generalized beta II distribution.

Author(s)

T. W. Yee and Kai Huang

References

Kleiber, C. and Kotz, S. (2003). *Statistical Size Distributions in Economics and Actuarial Sciences*, Hoboken, NJ, USA: Wiley-Interscience.

See Also

fisk, genbetaII.

Examples

```
fdata <- data.frame(y = rfisk(1000, shape = exp(1), scale = exp(2)))
fit <- vglm(y ~ 1, fisk(lss = FALSE), data = fdata, trace = TRUE)
coef(fit, matrix = TRUE)
Coef(fit)
```

---

fisk *Fisk Distribution family function*

Description

Maximum likelihood estimation of the 2-parameter Fisk distribution.

Usage

```
fisk(lscale = "loglink", lshape1.a = "loglink", iscale = NULL,
    ishape1.a = NULL, imethod = 1, lss = TRUE,
    gscale = exp(-5:5), gshape1.a = seq(0.75, 4, by = 0.25),
    probs.y = c(0.25, 0.5, 0.75), zero = "shape")
```

Arguments

`lss`	See `CommonVGAMffArguments` for important information.
`lshape1.a, lscale`	Parameter link functions applied to the (positive) parameters a and `scale`. See `Links` for more choices.
`iscale, ishape1.a, imethod, zero`	See `CommonVGAMffArguments` for information. For `imethod = 2` a good initial value for `iscale` is needed to obtain a good estimate for the other parameter.
`gscale, gshape1.a`	See `CommonVGAMffArguments` for information.
`probs.y`	See `CommonVGAMffArguments` for information.

Details

The 2-parameter Fisk (aka log-logistic) distribution is the 4-parameter generalized beta II distribution with shape parameter $q = p = 1$. It is also the 3-parameter Singh-Maddala distribution with shape parameter $q = 1$, as well as the Dagum distribution with $p = 1$. More details can be found in Kleiber and Kotz (2003).

The Fisk distribution has density

$$f(y) = ay^{a-1}/[b^a\{1 + (y/b)^a\}^2]$$

for $a > 0$, $b > 0$, $y \geq 0$. Here, b is the scale parameter `scale`, and a is a shape parameter. The cumulative distribution function is

$$F(y) = 1 - [1 + (y/b)^a]^{-1} = [1 + (y/b)^{-a}]^{-1}.$$

The mean is

$$E(Y) = b\,\Gamma(1 + 1/a)\,\Gamma(1 - 1/a)$$

provided $a > 1$; these are returned as the fitted values. This family function handles multiple responses.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Note

See the notes in `genbetaII`.

Author(s)

T. W. Yee

References

Kleiber, C. and Kotz, S. (2003). *Statistical Size Distributions in Economics and Actuarial Sciences*, Hoboken, NJ, USA: Wiley-Interscience.

See Also

Fisk, genbetaII, betaII, dagum, sinmad, inv.lomax, lomax, paralogistic, inv.paralogistic, simulate.vlm.

Examples

```
fdata <- data.frame(y = rfisk(200, shape = exp(1), exp(2)))
fit <- vglm(y ~ 1, fisk(lss = FALSE), data = fdata, trace = TRUE)
fit <- vglm(y ~ 1, fisk(ishape1.a = exp(2)), fdata, trace = TRUE)
coef(fit, matrix = TRUE)
Coef(fit)
summary(fit)
```

---

fittedvlm *Fitted Values of a VLM object*

Description

Extractor function for the fitted values of a model object that inherits from a *vector linear model* (VLM), e.g., a model of class "vglm".

Usage

```
fittedvlm(object, drop = FALSE, type.fitted = NULL,
          percentiles = NULL, ...)
```

Arguments

object	a model object that inherits from a VLM.
drop	Logical. If FALSE then the answer is a matrix. If TRUE then the answer is a vector.
type.fitted	Character. Some **VGAM** family functions have a type.fitted argument. If so then a different type of fitted value can be returned. It is recomputed from the model after convergence. Note: this is an experimental feature and not all **VGAM** family functions have this implemented yet. See CommonVGAMffArguments for more details.
percentiles	See CommonVGAMffArguments for details.
...	Currently unused.

Details

The "fitted values" usually corresponds to the mean response, however, because the **VGAM** package fits so many models, this sometimes refers to quantities such as quantiles. The mean may even not exist, e.g., for a Cauchy distribution.

Note that the fitted value is output from the @linkinv slot of the **VGAM** family function, where the eta argument is the $n \times M$ matrix of linear predictors.

Value

The fitted values evaluated at the final IRLS iteration.

Note

This function is one of several extractor functions for the **VGAM** package. Others include `coef`, `deviance`, `weights` and `constraints` etc. This function is equivalent to the methods function for the generic function `fitted.values`.

If `fit` is a VLM or VGLM then `fitted(fit)` and `predict(fit, type = "response")` should be equivalent (see `predictvglm`). The latter has the advantage in that it handles a `newdata` argument so that the fitted values can be computed for a different data set.

Author(s)

Thomas W. Yee

References

Chambers, J. M. and T. J. Hastie (eds) (1992). *Statistical Models in S*. Wadsworth & Brooks/Cole.

See Also

`fitted`, `predictvglm`, `vglmff-class`.

Examples

```
# Categorical regression example 1
pneumo <- transform(pneumo, let = log(exposure.time))
(fit1 <- vglm(cbind(normal, mild, severe) ~ let, propodds, pneumo))
fitted(fit1)

# LMS quantile regression example 2
fit2 <- vgam(BMI ~ s(age, df = c(4, 2)),
             lms.bcn(zero = 1), data = bmi.nz, trace = TRUE)
head(predict(fit2, type = "response"))  # Equals to both these:
head(fitted(fit2))
predict(fit2, type = "response", newdata = head(bmi.nz))

# Zero-inflated example 3
zdata <- data.frame(x2 = runif(nn <- 1000))
zdata <- transform(zdata,
                   pstr0.3  = logitlink(-0.5       , inverse = TRUE),
                   lambda.3 =   loglink(-0.5 + 2*x2, inverse = TRUE))
zdata <- transform(zdata,
         y1 = rzipois(nn, lambda = lambda.3, pstr0 = pstr0.3))
fit3 <- vglm(y1 ~ x2, zipoisson(zero = NULL), zdata, trace = TRUE)
head(fitted(fit3, type.fitted = "mean" ))  # E(Y) (the default)
head(fitted(fit3, type.fitted = "pobs0"))  # Pr(Y = 0)
head(fitted(fit3, type.fitted = "pstr0"))  # Prob of a structural 0
head(fitted(fit3, type.fitted = "onempstr0"))  # 1 - Pr(structural 0)
```

`fix.crossing` *Fixing a Quantile Regression having Crossing*

Description

Returns a similar object fitted with columns of the constraint matrices amalgamated so it is a partially parallel VGLM object. The columns combined correspond to certain crossing quantiles. This applies especially to an extlogF1() VGLM object.

Usage

```
fix.crossing.vglm(object, maxit = 100, trace = FALSE, ...)
```

Arguments

`object`	an object such as a `vglm` object with family function `extlogF1`.
`maxit, trace`	values for overwriting components in `vglm.control`. Setting these to `NULL` will mean the values in `vglm.control` on `object` will be retained.
`...`	additional optional arguments. Currently unused.

Details

The quantile crossing problem has been described as *disturbing* and *embarrassing*. This function was specifically written for a `vglm` with family function `extlogF1`. It examines the fitted quantiles of `object` to see if any cross. If so, then a pair of columns is combined to make those two quantiles parallel. After fitting the submodel it then repeats testing for crossing quantiles and repairing them, until there is no more quantile crossing detected. Note that it is possible that the quantiles cross in some subset of the covariate space not covered by the data—see `is.crossing`.

This function is fragile and likely to change in the future. For `extlogF1` models, it is assumed that argument `data` has been assigned a data frame, and that the default values of the argument `parallel` has been used; this means that the second constraint matrix is `diag(M)`. The constraint matrix of the intercept term remains unchanged as `diag(M)`.

Value

An object very similar to the original object, but with possibly different constraint matrices (partially parallel) so as to remove any quantile crossing.

See Also

`extlogF1`, `is.crossing`, `lms.bcn`. `vglm`.

Examples

```
## Not run:  ooo <- with(bmi.nz, order(age))
bmi.nz <- bmi.nz[ooo, ]  # Sort by age
with(bmi.nz, plot(age, BMI, col = "blue"))
mytau <- c(50, 93, 95, 97) / 100  # Some quantiles are quite close
fit1 <- vglm(BMI ~ ns(age, 7), extlogF1(mytau), bmi.nz, trace = TRUE)
plot(BMI ~ age, bmi.nz, col = "blue", las = 1,
     main = "Partially parallel (darkgreen) & nonparallel quantiles",
     sub = "Crossing quantiles are orange")
fix.crossing(fit1)
matlines(with(bmi.nz, age), fitted(fit1), lty = 1, col = "orange")
fit2 <- fix.crossing(fit1)  # Some quantiles have been fixed
constraints(fit2)
matlines(with(bmi.nz, age), fitted(fit2), lty = "dashed",
         col = "darkgreen", lwd = 2)
## End(Not run)
```

`flourbeetle` *Mortality of Flour Beetles from Carbon Disulphide*

Description

The `flourbeetle` data frame has 8 rows and 4 columns. Two columns are explanatory, the other two are responses.

Usage

```
data(flourbeetle)
```

Format

This data frame contains the following columns:

logdose `log10` applied to `CS2mgL`.

CS2mgL a numeric vector, the concentration of gaseous carbon disulphide in mg per litre.

exposed a numeric vector, counts; the number of beetles exposed to the poison.

killed a numeric vector, counts; the numbers killed.

Details

These data were originally given in Table IV of Bliss (1935) and are the combination of two series of toxicological experiments involving *Tribolium confusum*, also known as the flour beetle. Groups of such adult beetles were exposed for 5 hours of gaseous carbon disulphide at different concentrations, and their mortality measured.

Source

Bliss, C.I., 1935. The calculation of the dosage-mortality curve. *Annals of Applied Biology*, **22**, 134–167.

See Also

binomialff, probitlink.

Examples

```
fit1 <- vglm(cbind(killed, exposed - killed) ~ logdose,
             binomialff(link = probitlink), flourbeetle, trace = TRUE)
summary(fit1)
```

Foldnorm *The Folded-Normal Distribution*

Description

Density, distribution function, quantile function and random generation for the (generalized) folded-normal distribution.

Usage

```
dfoldnorm(x, mean = 0, sd = 1, a1 = 1, a2 = 1, log = FALSE)
pfoldnorm(q, mean = 0, sd = 1, a1 = 1, a2 = 1,
          lower.tail = TRUE, log.p = FALSE)
qfoldnorm(p, mean = 0, sd = 1, a1 = 1, a2 = 1,
          lower.tail = TRUE, log.p = FALSE, ...)
rfoldnorm(n, mean = 0, sd = 1, a1 = 1, a2 = 1)
```

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
n	number of observations. Same as rnorm.
mean, sd	see rnorm.
a1, a2	see foldnormal.
log	Logical. If TRUE then the log density is returned.
lower.tail, log.p	Same meaning as in pnorm or qnorm.
...	Arguments that can be passed into uniroot.

Details

See foldnormal, the **VGAM** family function for estimating the parameters, for the formula of the probability density function and other details.

Value

dfoldnorm gives the density, pfoldnorm gives the distribution function, qfoldnorm gives the quantile function, and rfoldnorm generates random deviates.

Author(s)

T. W. Yee and Kai Huang. Suggestions from Mauricio Romero led to improvements in `qfoldnorm()`.

See Also

`foldnormal`, `uniroot`.

Examples

```
## Not run:
m <- 1.5; SD <- exp(0)
x <- seq(-1, 4, len = 501)
plot(x, dfoldnorm(x, m = m, sd = SD), type = "l", ylim = 0:1,
     ylab = paste("foldnorm(m = ", m, ", sd = ",
                  round(SD, digits = 3), ")"), las = 1,
     main = "Blue is density, orange is CDF", col = "blue",
     sub = "Purple lines are the 10,20,...,90 percentiles")
abline(h = 0, col = "gray50")
lines(x, pfoldnorm(x, m = m, sd = SD), col = "orange")
probs <- seq(0.1, 0.9, by = 0.1)
Q <- qfoldnorm(probs, m = m, sd = SD)
lines(Q, dfoldnorm(Q, m, SD), col = "purple", lty = 3, type = "h")
lines(Q, pfoldnorm(Q, m, SD), col = "purple", lty = 3, type = "h")
abline(h = probs, col = "purple", lty = 3)
max(abs(pfoldnorm(Q, m = m, sd = SD) - probs))  # Should be 0

## End(Not run)
```

`foldnormal` *Folded Normal Distribution Family Function*

Description

Fits a (generalized) folded (univariate) normal distribution.

Usage

```
foldnormal(lmean = "identitylink", lsd = "loglink", imean = NULL,
   isd = NULL, a1 = 1, a2 = 1, nsimEIM = 500, imethod = 1,
   zero = NULL)
```

Arguments

`lmean, lsd`	Link functions for the mean and standard deviation parameters of the usual univariate normal distribution. They are μ and σ respectively. See `Links` for more choices.
`imean, isd`	Optional initial values for μ and σ. A `NULL` means a value is computed internally. See `CommonVGAMffArguments`.

a1, a2 Positive weights, called a_1 and a_2 below. Each must be of length 1.

nsimEIM, imethod, zero
See CommonVGAMffArguments.

Details

If a random variable has an ordinary univariate normal distribution then the absolute value of that random variable has an ordinary *folded normal distribution*. That is, the sign has not been recorded; only the magnitude has been measured.

More generally, suppose X is normal with mean mean and standard deviation sd. Let $Y = \max(a_1 X, -a_2 X)$ where a_1 and a_2 are positive weights. This means that $Y = a_1 X$ for $X > 0$, and $Y = a_2 X$ for $X < 0$. Then Y is said to have a *generalized folded normal distribution*. The ordinary folded normal distribution corresponds to the special case $a_1 = a_2 = 1$.

The probability density function of the ordinary folded normal distribution can be written `dnorm(y, mean, sd) + dnorm(y, -mean, sd)` for $y \geq 0$. By default, mean and log(sd) are the linear/additive predictors. Having mean=0 and sd=1 results in the *half-normal* distribution. The mean of an ordinary folded normal distribution is

$$E(Y) = \sigma \sqrt{2/\pi} \exp(-\mu^2/(2\sigma^2)) + \mu[1 - 2\Phi(-\mu/\sigma)]$$

and these are returned as the fitted values. Here, $\Phi()$ is the cumulative distribution function of a standard normal (pnorm).

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm and vgam.

Warning

Under- or over-flow may occur if the data is ill-conditioned. It is recommended that several different initial values be used to help avoid local solutions.

Note

The response variable for this family function is the same as uninormal except positive values are required. Reasonably good initial values are needed. Fisher scoring using simulation is implemented.

See CommonVGAMffArguments for general information about many of these arguments.

Yet to do: implement the results of Johnson (1962) which gives expressions for the EIM, albeit, under a different parameterization. Also, one element of the EIM appears to require numerical integration.

Author(s)

Thomas W. Yee

References

Lin, P. C. (2005). Application of the generalized folded-normal distribution to the process capability measures. *International Journal of Advanced Manufacturing Technology*, **26**, 825–830.

Johnson, N. L. (1962). The folded normal distribution: accuracy of estimation by maximum likelihood. *Technometrics*, **4**, 249–256.

See Also

`rfoldnorm`, `uninormal`, `dnorm`, `skewnormal`.

Examples

```
## Not run:  m <-  2; SD <- exp(1)
fdata <- data.frame(y = rfoldnorm(n <- 1000, m = m, sd = SD))
hist(with(fdata, y), prob = TRUE, main = paste("foldnormal(m = ",
     m, ", sd = ", round(SD, 2), ")"))
fit <- vglm(y ~ 1, foldnormal, data = fdata, trace = TRUE)
coef(fit, matrix = TRUE)
(Cfit <- Coef(fit))
# Add the fit to the histogram:
mygrid <- with(fdata, seq(min(y), max(y), len = 200))
lines(mygrid, dfoldnorm(mygrid, Cfit[1], Cfit[2]), col = "orange")

## End(Not run)
```

`formulavlm` *Model Formulae and Term Names for VGLMs*

Description

The methods function for `formula` to extract the formula from a fitted object, as well as a methods function to return the names of the terms in the formula.

Usage

```
## S3 method for class 'vlm'
formula(x, ...)
formulavlm(x, form.number = 1, ...)
term.names(model, ...)
term.namesvlm(model, form.number = 1, ...)
```

Arguments

`x, model`	A fitted model object.
`form.number`	Formula number, is 1 or 2. which correspond to the arguments `formula` and `form2` respectively.
`...`	Same as `formula`.

Details

The `formula` methods function is based on `formula`.

Value

The `formula` methods function should return something similar to `formula`. The `term.names` methods function should return a character string with the terms in the formula; this includes any intercept (which is denoted by `"(Intercept)"` as the first element.)

Author(s)

Thomas W. Yee

See Also

`has.interceptvlm`.

Examples

```
# Example: this is based on a glm example
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3, 1, 9); treatment <- gl(3, 3)
vglm.D93 <- vglm(counts ~ outcome + treatment, family = poissonff)
formula(vglm.D93)
pdata <- data.frame(counts, outcome, treatment)  # Better style
vglm.D93 <- vglm(counts ~ outcome + treatment, poissonff, data = pdata)
formula(vglm.D93)
term.names(vglm.D93)
responseName(vglm.D93)
has.intercept(vglm.D93)
```

Frank *Frank's Bivariate Distribution*

Description

Density, distribution function, and random generation for the (one parameter) bivariate Frank distribution.

Usage

```
dbifrankcop(x1, x2, apar, log = FALSE)
pbifrankcop(q1, q2, apar)
rbifrankcop(n, apar)
```

Arguments

x1, x2, q1, q2	vector of quantiles.
n	number of observations. Same as in runif.
apar	the positive association parameter.
log	Logical. If log = TRUE then the logarithm of the density is returned.

Details

See bifrankcop, the **VGAM** family functions for estimating the association parameter by maximum likelihood estimation, for the formula of the cumulative distribution function and other details.

Value

dbifrankcop gives the density, pbifrankcop gives the distribution function, and rbifrankcop generates random deviates (a two-column matrix).

Author(s)

T. W. Yee

References

Genest, C. (1987). Frank's family of bivariate distributions. *Biometrika*, **74**, 549–555.

See Also

bifrankcop.

Examples

```
## Not run: N <- 100; apar <- exp(2)
xx <- seq(-0.30, 1.30, len = N)
ox <- expand.grid(xx, xx)
zedd <- dbifrankcop(ox[, 1], ox[, 2], apar = apar)
contour(xx, xx, matrix(zedd, N, N))
zedd <- pbifrankcop(ox[, 1], ox[, 2], apar = apar)
contour(xx, xx, matrix(zedd, N, N))

plot(rr <- rbifrankcop(n = 3000, apar = exp(4)))
par(mfrow = c(1, 2))
hist(rr[, 1]); hist(rr[, 2])  # Should be uniform

## End(Not run)
```

Frechet *The Frechet Distribution*

Description

Density, distribution function, quantile function and random generation for the three parameter Frechet distribution.

Usage

```
dfrechet(x, location = 0, scale = 1, shape, log = FALSE)
pfrechet(q, location = 0, scale = 1, shape,
         lower.tail = TRUE, log.p = FALSE)
qfrechet(p, location = 0, scale = 1, shape,
         lower.tail = TRUE, log.p = FALSE)
rfrechet(n, location = 0, scale = 1, shape)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. Passed into `runif`.
`location, scale, shape`	the location parameter a, scale parameter b, and shape parameter s.
`log`	Logical. If `log = TRUE` then the logarithm of the density is returned.
`lower.tail, log.p`	Same meaning as in `punif` or `qunif`.

Details

See `frechet`, the **VGAM** family function for estimating the 2 parameters (without location parameter) by maximum likelihood estimation, for the formula of the probability density function and range restrictions on the parameters.

Value

`dfrechet` gives the density, `pfrechet` gives the distribution function, `qfrechet` gives the quantile function, and `rfrechet` generates random deviates.

Author(s)

T. W. Yee and Kai Huang

References

Castillo, E., Hadi, A. S., Balakrishnan, N. and Sarabia, J. S. (2005). *Extreme Value and Related Models with Applications in Engineering and Science*, Hoboken, NJ, USA: Wiley-Interscience.

See Also

frechet.

Examples

```
## Not run:  shape <- 5
x <- seq(-0.1, 3.5, length = 401)
plot(x, dfrechet(x, shape = shape), type = "l", ylab = "",
  main = "Frechet density divided into 10 equal areas",
  sub = "Orange = CDF", las = 1)
abline(h = 0, col = "blue", lty = 2)
qq <- qfrechet(seq(0.1, 0.9, by = 0.1), shape = shape)
lines(qq, dfrechet(qq, shape = shape), col = 2, lty = 2, type = "h")
lines(x, pfrechet(q = x, shape = shape), col = "orange")

## End(Not run)
```

---

frechet *Frechet Distribution Family Function*

Description

Maximum likelihood estimation of the 2-parameter Frechet distribution.

Usage

```
frechet(location = 0, lscale = "loglink",
  lshape = logofflink(offset = -2),
  iscale = NULL, ishape = NULL, nsimEIM = 250, zero = NULL)
```

Arguments

location	Numeric. Location parameter. It is called a below.
lscale, lshape	Link functions for the parameters; see Links for more choices.
iscale, ishape, zero, nsimEIM	See CommonVGAMffArguments for information.

Details

The (3-parameter) Frechet distribution has a density function that can be written

$$f(y) = \frac{sb}{(y-a)^2}[b/(y-a)]^{s-1} \exp[-(b/(y-a))^s]$$

for $y > a$ and scale parameter $b > 0$. The positive shape parameter is s. The cumulative distribution function is

$$F(y) = \exp[-(b/(y-a))^s].$$

The mean of Y is $a + b\Gamma(1 - 1/s)$ for $s > 1$ (these are returned as the fitted values). The variance of Y is $b^2[\Gamma(1 - 2/s) - \Gamma^2(1 - 1/s)]$ for $s > 2$.

Family `frechet` has a known, and $\log(b)$ and $\log(s - 2)$ are the default linear/additive predictors. The working weights are estimated by simulated Fisher scoring.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Warning

Family function `frechet` may fail for low values of the shape parameter, e.g., near 2 or lower.

Author(s)

T. W. Yee

References

Castillo, E., Hadi, A. S., Balakrishnan, N. and Sarabia, J. S. (2005). *Extreme Value and Related Models with Applications in Engineering and Science*, Hoboken, NJ, USA: Wiley-Interscience.

See Also

`rfrechet`, `gev`.

Examples

```
## Not run:
set.seed(123)
fdata <- data.frame(y1 = rfrechet(1000, shape = 2 + exp(1)))
with(fdata, hist(y1))
fit2 <- vglm(y1 ~ 1, frechet, data = fdata, trace = TRUE)
coef(fit2, matrix = TRUE)
Coef(fit2)
head(fitted(fit2))
with(fdata, mean(y1))
head(weights(fit2, type = "working"))
vcov(fit2)

## End(Not run)
```

freund61 *Freund's (1961) Bivariate Extension of the Exponential Distribution*

Description

Estimate the four parameters of the Freund (1961) bivariate extension of the exponential distribution by maximum likelihood estimation.

Usage

```
freund61(la = "loglink",  lap = "loglink",  lb = "loglink",
         lbp = "loglink", ia = NULL, iap = NULL, ib = NULL,
         ibp = NULL, independent = FALSE, zero = NULL)
```

Arguments

`la, lap, lb, lbp`
: Link functions applied to the (positive) parameters α, α', β and β', respectively (the "p" stands for "prime"). See `Links` for more choices.

`ia, iap, ib, ibp`
: Initial value for the four parameters respectively. The default is to estimate them all internally.

`independent`
: Logical. If `TRUE` then the parameters are constrained to satisfy $\alpha = \alpha'$ and $\beta = \beta'$, which implies that y_1 and y_2 are independent and each have an ordinary exponential distribution.

`zero`
: A vector specifying which linear/additive predictors are modelled as intercepts only. The values can be from the set {1,2,3,4}. The default is none of them. See `CommonVGAMffArguments` for more information.

Details

This model represents one type of bivariate extension of the exponential distribution that is applicable to certain problems, in particular, to two-component systems which can function if one of the components has failed. For example, engine failures in two-engine planes, paired organs such as peoples' eyes, ears and kidneys. Suppose y_1 and y_2 are random variables representing the lifetimes of two components A and B in a two component system. The dependence between y_1 and y_2 is essentially such that the failure of the B component changes the parameter of the exponential life distribution of the A component from α to α', while the failure of the A component changes the parameter of the exponential life distribution of the B component from β to β'.

The joint probability density function is given by

$$f(y_1, y_2) = \alpha\beta' \exp(-\beta' y_2 - (\alpha + \beta - \beta')y_1)$$

for $0 < y_1 < y_2$, and

$$f(y_1, y_2) = \beta\alpha' \exp(-\alpha' y_1 - (\alpha + \beta - \alpha')y_2)$$

for $0 < y_2 < y_1$. Here, all four parameters are positive, as well as the responses y_1 and y_2. Under this model, the probability that component A is the first to fail is $\alpha/(\alpha + \beta)$. The time to the first failure is distributed as an exponential distribution with rate $\alpha + \beta$. Furthermore, the distribution of the time from first failure to failure of the other component is a mixture of Exponential(α') and Exponential(β') with proportions $\beta/(\alpha + \beta)$ and $\alpha/(\alpha + \beta)$ respectively.

The marginal distributions are, in general, not exponential. By default, the linear/additive predictors are $\eta_1 = \log(\alpha)$, $\eta_2 = \log(\alpha')$, $\eta_3 = \log(\beta)$, $\eta_4 = \log(\beta')$.

A special case is when $\alpha = \alpha'$ and $\beta = \beta'$, which means that y_1 and y_2 are independent, and both have an ordinary exponential distribution with means $1/\alpha$ and $1/\beta$ respectively.

Fisher scoring is used, and the initial values correspond to the MLEs of an intercept model. Consequently, convergence may take only one iteration.

Value

An object of class "vglmff" (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Note

To estimate all four parameters, it is necessary to have some data where $y_1 < y_2$ and $y_2 < y_1$.

The response must be a two-column matrix, with columns y_1 and y_2. Currently, the fitted value is a matrix with two columns; the first column has values $(\alpha' + \beta)/(\alpha'(\alpha + \beta))$ for the mean of y_1, while the second column has values $(\beta' + \alpha)/(\beta'(\alpha + \beta))$ for the mean of y_2. The variance of y_1 is

$$\frac{(\alpha')^2 + 2\alpha\beta + \beta^2}{(\alpha')^2(\alpha + \beta)^2},$$

the variance of y_2 is

$$\frac{(\beta')^2 + 2\alpha\beta + \alpha^2}{(\beta')^2(\alpha + \beta)^2},$$

the covariance of y_1 and y_2 is

$$\frac{\alpha'\beta' - \alpha\beta}{\alpha'\beta'(\alpha + \beta)^2}.$$

Author(s)

T. W. Yee

References

Freund, J. E. (1961). A bivariate extension of the exponential distribution. *Journal of the American Statistical Association*, **56**, 971–977.

See Also

`exponential`.

Examples

```
fdata <- data.frame(y1 = rexp(nn <- 1000, rate = exp(1)))
fdata <- transform(fdata, y2 = rexp(nn, rate = exp(2)))
fit1 <- vglm(cbind(y1, y2) ~ 1, freund61, fdata, trace = TRUE)
coef(fit1, matrix = TRUE)
Coef(fit1)
vcov(fit1)
head(fitted(fit1))
summary(fit1)

# y1 and y2 are independent, so fit an independence model
fit2 <- vglm(cbind(y1, y2) ~ 1, freund61(indep = TRUE),
             data = fdata, trace = TRUE)
coef(fit2, matrix = TRUE)
constraints(fit2)
pchisq(2 * (logLik(fit1) - logLik(fit2)),  # p-value
       df = df.residual(fit2) - df.residual(fit1),
       lower.tail = FALSE)
lrtest(fit1, fit2)  # Better alternative
```

Gaitdbinom   *Generally Altered, Inflated, Truncated and Deflated Binomial Distribution*

Description

Density, distribution function, quantile function and random generation for the generally altered, inflated, truncated and deflated binomial distribution. Both parametric and nonparametric variants are supported; these are based on finite mixtures of the parent with itself and the multinomial logit model (MLM) respectively.

Usage

```
dgaitdbinom(x, size.p, prob.p,
            a.mix = NULL, a.mlm = NULL,
            i.mix = NULL, i.mlm = NULL,
            d.mix = NULL, d.mlm = NULL, truncate = NULL,
            pobs.mix = 0, pobs.mlm = 0,
            pstr.mix = 0, pstr.mlm = 0,
            pdip.mix = 0, pdip.mlm = 0, byrow.aid = FALSE,
            size.a = size.p, size.i = size.p, size.d = size.p,
            prob.a = prob.p, prob.i = prob.p, prob.d = prob.p,
            log = FALSE, ...)
pgaitdbinom(q, size.p, prob.p,
            a.mix = NULL, a.mlm = NULL,
            i.mix = NULL, i.mlm = NULL,
            d.mix = NULL, d.mlm = NULL, truncate = NULL,
            pobs.mix = 0, pobs.mlm = 0,
```

```
              pstr.mix = 0, pstr.mlm = 0,
              pdip.mix = 0, pdip.mlm = 0, byrow.aid = FALSE,
              size.a = size.p, size.i = size.p, size.d = size.p,
              prob.a = prob.p, prob.i = prob.p, prob.d = prob.p,
              lower.tail = TRUE, ...)
qgaitdbinom(p, size.p, prob.p,
              a.mix = NULL, a.mlm = NULL,
              i.mix = NULL, i.mlm = NULL,
              d.mix = NULL, d.mlm = NULL, truncate = NULL,
              pobs.mix = 0, pobs.mlm = 0,
              pstr.mix = 0, pstr.mlm = 0,
              pdip.mix = 0, pdip.mlm = 0, byrow.aid = FALSE,
              size.a = size.p, size.i = size.p, size.d = size.p,
              prob.a = prob.p, prob.i = prob.p, prob.d = prob.p, ...)
rgaitdbinom(n, size.p, prob.p,
              a.mix = NULL, a.mlm = NULL,
              i.mix = NULL, i.mlm = NULL,
              d.mix = NULL, d.mlm = NULL, truncate = NULL,
              pobs.mix = 0, pobs.mlm = 0,
              pstr.mix = 0, pstr.mlm = 0,
              pdip.mix = 0, pdip.mlm = 0, byrow.aid = FALSE,
              size.a = size.p, size.i = size.p, size.d = size.p,
              prob.a = prob.p, prob.i = prob.p, prob.d = prob.p, ...)
```

Arguments

`x, q, p, n, log, lower.tail`
: Same meaning as in `Binomial`.

`size.p, prob.p`
: Same meaning as in `Binomial`. See `Gaitdpois` for generic information.

`size.a, prob.a`
: See `Gaitdpois` for generic information.

`size.i, prob.i`
: See `Gaitdpois` for generic information.

`size.d, prob.d`
: See `Gaitdpois` for generic information.

`truncate`
: See `Gaitdpois` for generic information.

`a.mix, i.mix, d.mix`
: See `Gaitdpois` for generic information.

`a.mlm, i.mlm, d.mlm`
: See `Gaitdpois` for generic information.

`pstr.mix, pstr.mlm, byrow.aid`
: See `Gaitdpois` for generic information.

`pobs.mix, pobs.mlm`
: See `Gaitdpois` for generic information.

`pdip.mix, pdip.mlm`
: See `Gaitdpois` for generic information.

`...`
: Arguments such as `max.support` that are ignored. This will occur internally within `dgaitdplot`.

Details

These functions for the GAITD binomial distribution are analogous to the GAITD Poisson, hence most details have been put in Gaitdpois.

Value

dgaitdbinom gives the density, pgaitdbinom gives the distribution function, qgaitdbinom gives the quantile function, and rgaitdbinom generates random deviates. The default values of the arguments correspond to ordinary dbinom, pbinom, qbinom, rbinom respectively.

Warning

See Gaitdpois about the dangers of too much inflation and/or deflation on GAITD PMFs, and the difficulties detecting such.

Note

Functions Posbinom have been moved to **VGAMdata**. It is better to use dgaitdbinom(x, size, prob, truncate = 0) instead of dposbinom(x, size, prob), etc.

Author(s)

T. W. Yee.

See Also

Gaitdpois, Gaitdnbinom, multinomial, Gaitdlog, Gaitdzeta.

Examples

```
 size <- 20
ivec <- c(6, 10); avec <- c(8, 11); prob <- 0.25; xgrid <- 0:25
tvec <- 14; pobs.a <- 0.05; pstr.i <- 0.15
dvec <- 5; pdip.mlm <- 0.05
(ddd <- dgaitdbinom(xgrid, size, prob.p = prob,
   prob.a = prob + 0.05, truncate = tvec, pobs.mix = pobs.a,
   pdip.mlm = pdip.mlm, d.mlm = dvec,
   pobs.mlm = pobs.a, a.mlm = avec,
   pstr.mix = pstr.i, i.mix = ivec))
## Not run:  dgaitdplot(c(size, prob), ylab = "Probability",
   xlab = "x", pobs.mix = pobs.mix,
   pobs.mlm = pobs.a, a.mlm = avec, all.lwd = 3,
   pdip.mlm = pdip.mlm, d.mlm = dvec, fam = "binom",
   pstr.mix = pstr.i, i.mix = ivec, deflation = TRUE,
   main = "GAITD Combo PMF---Binomial Parent")
## End(Not run)
```

Gaitdlog *Generally Altered, Inflated, Truncated and Deflated Logarithmic Distribution*

Description

Density, distribution function, quantile function and random generation for the generally altered, inflated, truncated and deflated logarithmic distribution. Both parametric and nonparametric variants are supported; these are based on finite mixtures of the parent with itself and the multinomial logit model (MLM) respectively.

Usage

```
dgaitdlog(x, shape.p, a.mix = NULL, a.mlm = NULL,
          i.mix = NULL, i.mlm = NULL,
          d.mix = NULL, d.mlm = NULL, truncate = NULL,
          max.support = Inf, pobs.mix = 0, pobs.mlm = 0,
          pstr.mix = 0, pstr.mlm = 0,
          pdip.mix = 0, pdip.mlm = 0, byrow.aid = FALSE,
          shape.a = shape.p, shape.i = shape.p, shape.d = shape.p,
          log = FALSE)
pgaitdlog(q, shape.p, a.mix = NULL, a.mlm = NULL,
          i.mix = NULL, i.mlm = NULL,
          d.mix = NULL, d.mlm = NULL, truncate = NULL,
          max.support = Inf, pobs.mix = 0, pobs.mlm = 0,
          pstr.mix = 0, pstr.mlm = 0,
          pdip.mix = 0, pdip.mlm = 0, byrow.aid = FALSE,
          shape.a = shape.p, shape.i = shape.p, shape.d = shape.p,
          lower.tail = TRUE)
qgaitdlog(p, shape.p, a.mix = NULL, a.mlm = NULL,
          i.mix = NULL, i.mlm = NULL,
          d.mix = NULL, d.mlm = NULL, truncate = NULL,
          max.support = Inf, pobs.mix = 0, pobs.mlm = 0,
          pstr.mix = 0, pstr.mlm = 0,
          pdip.mix = 0, pdip.mlm = 0, byrow.aid = FALSE,
          shape.a = shape.p, shape.i = shape.p, shape.d = shape.p)
rgaitdlog(n, shape.p, a.mix = NULL, a.mlm = NULL,
          i.mix = NULL, i.mlm = NULL,
          d.mix = NULL, d.mlm = NULL, truncate = NULL,
          max.support = Inf, pobs.mix = 0, pobs.mlm = 0,
          pstr.mix = 0, pstr.mlm = 0,
          pdip.mix = 0, pdip.mlm = 0, byrow.aid = FALSE,
          shape.a = shape.p, shape.i = shape.p, shape.d = shape.p)
```

Arguments

`x, q, p, n, log, lower.tail`
: Same meaning as in `dlog`.

`shape.p, shape.a, shape.i, shape.d`
: Same meaning as shape for dlog, i.e., for an ordinary logarithmic distribution. See Gaitdpois for generic information.

`truncate, max.support`
: See Gaitdpois for generic information.

`a.mix, i.mix, d.mix`
: See Gaitdpois for generic information.

`a.mlm, i.mlm, d.mlm`
: See Gaitdpois for generic information.

`pobs.mlm, pstr.mlm, pdip.mlm, byrow.aid`
: See Gaitdpois for generic information.

`pobs.mix, pstr.mix, pdip.mix`
: See Gaitdpois for generic information.

Details

These functions for the logarithmic distribution are analogous to the Poisson, hence most details have been put in Gaitdpois. These functions do what Oalog, Oilog, Otlog collectively did plus much more.

Value

dgaitdlog gives the density, pgaitdlog gives the distribution function, qgaitdlog gives the quantile function, and rgaitdlog generates random deviates. The default values of the arguments correspond to ordinary dlog, plog, qlog, rlog respectively.

Warning

See Gaitdpois about the dangers of too much inflation and/or deflation on GAITD PMFs, and the difficulties detecting such.

Note

See Gaitdpois for general information also relevant to this parent distribution.

Author(s)

T. W. Yee.

See Also

gaitdlog, Gaitdpois, dgaitdplot, Gaitdzeta, multinomial, Oalog, Oilog, Otlog.

Examples

```
ivec <- c(2, 10); avec <- ivec + 1; shape <- 0.995; xgrid <- 0:15
max.support <- 15; pobs.a <- 0.10; pstr.i <- 0.15
dvec <- 1; pdip.mlm <- 0.05
(ddd <- dgaitdlog(xgrid, shape,
   max.support = max.support, pobs.mix = pobs.a,
```

```
    pdip.mlm = pdip.mlm, d.mlm = dvec,
    a.mix = avec, pstr.mix = pstr.i, i.mix = ivec))
## Not run:  dgaitdplot(shape, ylab = "Probability", xlab = "x",
    max.support = max.support, pobs.mix = 0,
    pobs.mlm = 0, a.mlm = avec, all.lwd = 3,
    pdip.mlm = pdip.mlm, d.mlm = dvec, fam = "log",
    pstr.mix = pstr.i, i.mix = ivec, deflation = TRUE,
    main = "GAITD Combo PMF---Logarithmic Parent")
## End(Not run)
```

gaitdlog *Generally Altered, Inflated, Truncated and Deflated Logarithmic Regression*

Description

Fits a generally altered, inflated, truncated and deflated logarithmic regression by MLE. The GAITD combo model having 7 types of special values is implemented. This allows logarithmic mixtures on nested and/or partitioned support as well as a multinomial logit model for altered, inflated and deflated values. Truncation may include the upper tail.

Usage

```
gaitdlog(a.mix = NULL, i.mix = NULL, d.mix = NULL,
         a.mlm = NULL, i.mlm = NULL, d.mlm = NULL,
         truncate = NULL, max.support = Inf,
         zero = c("pobs", "pstr", "pdip"), eq.ap = TRUE, eq.ip = TRUE,
         eq.dp = TRUE, parallel.a = FALSE,
         parallel.i = FALSE, parallel.d = FALSE,
         lshape.p = "logitlink", lshape.a = lshape.p,
         lshape.i = lshape.p, lshape.d = lshape.p,
         type.fitted = c("mean", "shapes", "pobs.mlm", "pstr.mlm",
         "pdip.mlm", "pobs.mix", "pstr.mix", "pdip.mix", "Pobs.mix",
         "Pstr.mix", "Pdip.mix", "nonspecial",
         "Numer", "Denom.p", "sum.mlm.i", "sum.mix.i", "sum.mlm.d",
         "sum.mix.d", "ptrunc.p", "cdf.max.s"),
         gshape.p = -expm1(-7 * ppoints(12)), gpstr.mix = ppoints(7) / 3,
         gpstr.mlm = ppoints(7) / (3 + length(i.mlm)),
         imethod = 1, mux.init = c(0.75, 0.5, 0.75),
         ishape.p = NULL, ishape.a = ishape.p,
         ishape.i = ishape.p, ishape.d = ishape.p,
         ipobs.mix = NULL, ipstr.mix = NULL, ipdip.mix = NULL,
         ipobs.mlm = NULL, ipstr.mlm = NULL, ipdip.mlm = NULL,
         byrow.aid = FALSE, ishrinkage = 0.95, probs.y = 0.35)
```

Arguments

`truncate, max.support`
: See gaitdpoisson.

`a.mix, i.mix, d.mix`
: See gaitdpoisson.

`a.mlm, i.mlm, d.mlm`
: See gaitdpoisson.

`lshape.p, lshape.a, lshape.i, lshape.d`
: Link functions. See gaitdpoisson and Links for more choices and information. Actually, it is usually a good idea to set these arguments equal to logffMlink because the log-mean is the first linear/additive predictor so it is like a Poisson regression.

`eq.ap, eq.ip, eq.dp`
: Single logical each. See gaitdpoisson.

`parallel.a, parallel.i, parallel.d`
: Single logical each. See gaitdpoisson.

`type.fitted, mux.init`
: See gaitdpoisson.

`imethod, ipobs.mix, ipstr.mix, ipdip.mix`
: See CommonVGAMffArguments and gaitdpoisson for information.

`ipobs.mlm, ipstr.mlm, ipdip.mlm, byrow.aid`
: See CommonVGAMffArguments and gaitdpoisson for information.

`gpstr.mix, gpstr.mlm`
: See CommonVGAMffArguments and gaitdpoisson for information.

`gshape.p, ishape.p`
: See CommonVGAMffArguments and gaitdpoisson for information. The former argument is used only if the latter is not given. Practical experience has shown that good initial values are needed, so if convergence is not obtained then try a finer grid.

`ishape.a, ishape.i, ishape.d`
: See CommonVGAMffArguments and gaitdpoisson for information.

`probs.y, ishrinkage`
: See CommonVGAMffArguments and gaitdpoisson for information.

`zero`
: See gaitdpoisson and CommonVGAMffArguments for information.

Details

Many details to this family function can be found in gaitdpoisson because it is also a 1-parameter discrete distribution. This function currently does not handle multiple responses. Further details are at Gaitdlog.

As alluded to above, when there are covariates it is much more interpretable to model the mean rather than the shape parameter. Hence logffMlink is recommended. (This might become the default in the future.) So installing **VGAMextra** is a good idea.

Apart from the order of the linear/additive predictors, the following are (or should be) equivalent: `gaitdlog()` and `logff()`, `gaitdlog(a.mix = 1)` and `oalog(zero = "pobs1")`, `gaitdlog(i.mix = 1)` and `oilog(zero = "pstr1")`, `gaitdlog(truncate = 1)` and `otlog()`. The functions oalog, oilog and otlog have been placed in **VGAMdata**.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, rrvglm and vgam.

Warning

See gaitdpoisson.

Note

See gaitdpoisson.

Author(s)

T. W. Yee

See Also

Gaitdlog, logff, logffMlink, Gaitdpois, gaitdpoisson, gaitdzeta, spikeplot, goffset, Trunc, oalog, oilog, otlog, CommonVGAMffArguments, rootogram4, simulate.vlm.

Examples

```
avec <- c(5, 10)  # Alter these values parametrically
ivec <- c(3, 15)  # Inflate these values
tvec <- c(6, 7)   # Truncate these values
max.support <- 20; set.seed(1)
pobs.a <- pstr.i <- 0.1
gdata <- data.frame(x2 = runif(nn <- 1000))
gdata <- transform(gdata, shape.p = logitlink(2+0.5*x2, inverse = TRUE))
gdata <- transform(gdata,
  y1 = rgaitdlog(nn, shape.p, a.mix = avec, pobs.mix = pobs.a,
                i.mix = ivec, pstr.mix = pstr.i, truncate = tvec,
                max.support = max.support))
gaitdlog(a.mix = avec, i.mix = ivec, max.support = max.support)
with(gdata, table(y1))
## Not run:  spikeplot(with(gdata, y1), las = 1)
fit7 <- vglm(y1 ~ x2, trace = TRUE, data = gdata,
             gaitdlog(i.mix = ivec, truncate = tvec,
                      max.support = max.support, a.mix = avec,
                      eq.ap = TRUE, eq.ip = TRUE))
head(fitted(fit7, type.fitted = "Pstr.mix"))
head(predict(fit7))
t(coef(fit7, matrix = TRUE))  # Easier to see with t()
summary(fit7)
## Not run:  spikeplot(with(gdata, y1), lwd = 2, ylim = c(0, 0.4))
plotdgaitd(fit7, new.plot = FALSE, offset.x = 0.2, all.lwd = 2)
## End(Not run)
```

Gaitdnbinom *Generally Altered, Inflated, Truncated and Deflated Negative Binomial Distribution*

Description

Density, distribution function, quantile function and random generation for the generally altered, inflated, truncated and deflated negative binomial (GAITD-NB) distribution. Both parametric and nonparametric variants are supported; these are based on finite mixtures of the parent with itself and the multinomial logit model (MLM) respectively.

Usage

```
dgaitdnbinom(x, size.p, munb.p,
             a.mix = NULL, a.mlm = NULL,
             i.mix = NULL, i.mlm = NULL,
             d.mix = NULL, d.mlm = NULL, truncate = NULL,
             max.support = Inf, pobs.mix = 0, pobs.mlm = 0,
             pstr.mix = 0, pstr.mlm = 0,
             pdip.mix = 0, pdip.mlm = 0, byrow.aid = FALSE,
             size.a = size.p, size.i = size.p, size.d = size.p,
             munb.a = munb.p, munb.i = munb.p, munb.d = munb.p,
             log = FALSE)
pgaitdnbinom(q, size.p, munb.p,
             a.mix = NULL, a.mlm = NULL,
             i.mix = NULL, i.mlm = NULL,
             d.mix = NULL, d.mlm = NULL, truncate = NULL,
             max.support = Inf, pobs.mix = 0, pobs.mlm = 0,
             pstr.mix = 0, pstr.mlm = 0,
             pdip.mix = 0, pdip.mlm = 0, byrow.aid = FALSE,
             size.a = size.p, size.i = size.p, size.d = size.p,
             munb.a = munb.p, munb.i = munb.p, munb.d = munb.p,
             lower.tail = TRUE)
qgaitdnbinom(p, size.p, munb.p,
             a.mix = NULL, a.mlm = NULL,
             i.mix = NULL, i.mlm = NULL,
             d.mix = NULL, d.mlm = NULL, truncate = NULL,
             max.support = Inf, pobs.mix = 0, pobs.mlm = 0,
             pstr.mix = 0, pstr.mlm = 0,
             pdip.mix = 0, pdip.mlm = 0, byrow.aid = FALSE,
             size.a = size.p, size.i = size.p, size.d = size.p,
             munb.a = munb.p, munb.i = munb.p, munb.d = munb.p)
rgaitdnbinom(n, size.p, munb.p,
             a.mix = NULL, a.mlm = NULL,
             i.mix = NULL, i.mlm = NULL,
             d.mix = NULL, d.mlm = NULL, truncate = NULL,
             max.support = Inf, pobs.mix = 0, pobs.mlm = 0,
```

```
pstr.mix = 0, pstr.mlm = 0,
pdip.mix = 0, pdip.mlm = 0, byrow.aid = FALSE,
size.a = size.p, size.i = size.p, size.d = size.p,
munb.a = munb.p, munb.i = munb.p, munb.d = munb.p)
```

Arguments

`x, q, p, n, log, lower.tail`
: Same meaning as in `rnbinom`.

`size.p, munb.p`
: Same meaning as in `rnbinom`. See `Gaitdpois` for generic information.

`size.a, munb.a`
: See `Gaitdpois` for generic information.

`size.i, munb.i`
: See `Gaitdpois` for generic information.

`size.d, munb.d`
: See `Gaitdpois` for generic information.

`truncate, max.support`
: See `Gaitdpois` for generic information.

`a.mix, i.mix, d.mix`
: See `Gaitdpois` for generic information.

`a.mlm, i.mlm, d.mlm`
: See `Gaitdpois` for generic information.

`pobs.mlm, pstr.mlm, byrow.aid`
: See `Gaitdpois` for generic information.

`pobs.mix, pstr.mix`
: See `Gaitdpois` for generic information.

`pdip.mix, pdip.mlm`
: See `Gaitdpois` for generic information.

Details

These functions for the NBD are analogous to the Poisson, hence most details have been put in `Gaitdpois`. The NBD has two possible parameterizations: one involving a probability (argument begins with `prob`) and the other the mean (beginning with `mu`). Only the latter is supported here.

For now, arguments such as `prob.p` and `prob.a` are no longer supported. That's because `mu` is more likely to be used by most statisticians than `prob`; see `dnbinom`.

Value

`dgaitdnbinom` gives the density, `pgaitdnbinom` gives the distribution function, `qgaitdnbinom` gives the quantile function, and `rgaitdnbinom` generates random deviates. The default values of the arguments correspond to ordinary `dnbinom`, `pnbinom`, `qnbinom`, `rnbinom` respectively.

Warning

See `Gaitdpois` about the dangers of too much inflation and/or deflation on GAITD PMFs, and the difficulties detecting such.

Note

Four functions were moved from **VGAM** to **VGAMdata**; they can be seen at Posnegbin. It is preferable to use `dgaitdnbinom(x, size, munb.p = munb, truncate = 0)` instead of `dposnbinom(x, size, munb = munb)`, etc.

Author(s)

T. W. Yee.

See Also

gaitdnbinomial, Gaitdpois, multinomial, Gaitdbinom, Gaitdlog, Gaitdzeta.

Examples

```
size <- 10; xgrid <- 0:25
ivec <- c(5, 6, 10, 14); avec <- c(8, 11); munb <- 10
tvec <- 15; pobs.a <- 0.05; pstr.i <- 0.25
dvec <- 13; pdip.mlm <- 0.03; pobs.mlm <- 0.05
(ddd <- dgaitdnbinom(xgrid, size, munb.p = munb, munb.a = munb + 5,
   truncate = tvec, pobs.mix = pobs.a,
   pdip.mlm = pdip.mlm, d.mlm = dvec,
   pobs.mlm = pobs.a, a.mlm = avec,
   pstr.mix = pstr.i, i.mix = ivec))
## Not run: dgaitdplot(c(size, munb), fam = "nbinom",
  ylab = "Probability", xlab = "x", xlim = c(0, 25),
  truncate = tvec, pobs.mix = pobs.mix,
  pobs.mlm = pobs.mlm, a.mlm = avec, all.lwd = 3,
  pdip.mlm = pdip.mlm, d.mlm = dvec,
  pstr.mix = pstr.i, i.mix = ivec, deflation = TRUE,
  main = "GAITD Combo PMF---NB Parent")
## End(Not run)
```

`gaitdnbinomial` *Generally Altered, Inflated, Truncated and Deflated Negative Binomial Regression*

Description

Fits a generally altered, inflated truncated and deflated negative binomial regression by MLE. The GAITD combo model having 7 types of special values is implemented. This allows mixtures of negative binomial distributions on nested and/or partitioned support as well as a multinomial logit model for (nonparametric) altered, inflated and deflated values.

Usage

```
gaitdnbinomial(a.mix = NULL, i.mix = NULL, d.mix = NULL,
    a.mlm = NULL, i.mlm = NULL, d.mlm = NULL,
    truncate = NULL, zero = c("size", "pobs", "pstr", "pdip"),
    eq.ap = TRUE, eq.ip = TRUE, eq.dp = TRUE,
    parallel.a = FALSE, parallel.i = FALSE, parallel.d = FALSE,
    lmunb.p = "loglink",
    lmunb.a = lmunb.p, lmunb.i = lmunb.p, lmunb.d = lmunb.p,
    lsize.p = "loglink",
    lsize.a = lsize.p, lsize.i = lsize.p, lsize.d = lsize.p,
    type.fitted = c("mean", "munbs", "sizes", "pobs.mlm",
    "pstr.mlm", "pdip.mlm", "pobs.mix", "pstr.mix", "pdip.mix",
    "Pobs.mix", "Pstr.mix", "Pdip.mix", "nonspecial", "Numer",
    "Denom.p", "sum.mlm.i", "sum.mix.i",
    "sum.mlm.d", "sum.mix.d", "ptrunc.p", "cdf.max.s"),
    gpstr.mix = ppoints(7) / 3,
    gpstr.mlm = ppoints(7) / (3 + length(i.mlm)),
    imethod = 1, mux.init = c(0.75, 0.5, 0.75, 0.5),
    imunb.p = NULL, imunb.a = imunb.p,
    imunb.i = imunb.p, imunb.d = imunb.p,
    isize.p = NULL,  isize.a = isize.p,
    isize.i = isize.p, isize.d = isize.p,
    ipobs.mix = NULL, ipstr.mix = NULL,
    ipdip.mix = NULL, ipobs.mlm = NULL,
    ipstr.mlm = NULL, ipdip.mlm = NULL,
    byrow.aid = FALSE, ishrinkage = 0.95, probs.y = 0.35,
    nsimEIM = 500, cutoff.prob = 0.999, eps.trig = 1e-7,
    nbd.max.support = 4000, max.chunk.MB = 30)
```

Arguments

`truncate` See `gaitdpoisson`.

`a.mix, i.mix, d.mix`
: See `gaitdpoisson`.

`a.mlm, i.mlm, d.mlm`
: See `gaitdpoisson`.

`lmunb.p, lmunb.a, lmunb.i, lmunb.d`
: Link functions pertaining to the mean parameters. See `gaitdpoisson` where `llambda.p` etc. are the equivalent.

`lsize.p, lsize.a, lsize.i, lsize.d`
: Link functions pertaining to the `size` parameters. See `NegBinomial`.

`eq.ap, eq.ip, eq.dp`
: See `gaitdpoisson`. These apply to both `munb` and `size` parameters simultaneously. See `NegBinomial` also.

`parallel.a, parallel.i, parallel.d`
: See `gaitdpoisson`.

`type.fitted` See `gaitdpoisson`.

`gpstr.mix, gpstr.mlm`
: See `gaitdpoisson`.

`imethod, ipobs.mix, ipstr.mix, ipdip.mix`
: See `gaitdpoisson` and `CommonVGAMffArguments`.

`ipobs.mlm, ipstr.mlm, ipdip.mlm`
: See `gaitdpoisson`.

`mux.init`
: Numeric, of length 4. General downward multiplier for initial values for the sample proportions (MLEs actually). See `gaitdpoisson`. The fourth value corresponds to `size`.

`imunb.p, imunb.a, imunb.i, imunb.d`
: See `gaitdpoisson`; `imunb.p` is similar to `ilambda.p`, etc.

`isize.p, isize.a, isize.i, isize.d`
: See `gaitdpoisson`; `isize.p` is similar to `ilambda.p`, etc.

`probs.y, ishrinkage`
: See `CommonVGAMffArguments` for information.

`byrow.aid`
: Details are at `Gaitdpois`.

`zero`
: See `gaitdpoisson` and `CommonVGAMffArguments`.

`nsimEIM, cutoff.prob, eps.trig`
: See `negbinomial`.

`nbd.max.support, max.chunk.MB`
: See `negbinomial`.

Details

The GAITD–NB combo model is the pinnacle of GAITD regression for counts because it potentially handles underdispersion, equidispersion and overdispersion relative to the Poisson, as well as alteration, inflation, deflation and truncation at arbitrary support points. In contrast, `gaitdpoisson` cannot handle overdispersion so well. The GAITD–NB is so flexible that it can accommodate up to seven modes.

The full GAITD–NB–NB–MLM–NB-MLM–NB-MLM combo model may be fitted with this family function. There are seven types of special values and all arguments for these may be used in a single model. Here, the MLM represents the nonparametric while the NB refers to the negative binomial mixtures. The defaults for this function correspond to an ordinary negative binomial regression so that `negbinomial` is called instead.

While much of the documentation here draws upon `gaitdpoisson`, there are additional details here because the NBD is a *two* parameter distribution that handles *overdispersion* relative to the Possion. Consequently, this family function is exceeding flexible and there are many more pitfalls to avoid.

The order of the linear/additive predictors is best explained by an example. Suppose a combo model has `length(a.mix) > 3` and `length(i.mix) > 3`, `length(d.mix) > 3`, `a.mlm = 3:5`, `i.mlm = 6:9` and `d.mlm = 10:12`, say. Then `loglink(munb.p)` and `loglink(size.p)` are the first two. The third is `multilogitlink(pobs.mix)` followed by `loglink(munb.a)` and `loglink(size.a)` because `a.mix` is long enough. The sixth is `multilogitlink(pstr.mix)` followed by `loglink(munb.i)` and `loglink(size.i)` because `i.mix` is long enough. The ninth is `multilogitlink(pdip.mix)` followed by `loglink(munb.d)` and `loglink(size.d)` because `d.mix` is long enough. Next are the probabilities for the `a.mlm` values. Then are the probabilities for the `i.mlm` values. Lastly are the probabilities for the `d.mlm` values. All the probabilities are estimated by one big MLM and effectively the "(Others)" column of left over probabilities is associated with the nonspecial values. These might be called the *nonspecial baseline probabilities* (NBP) or reserve probabilities. The dimension of the vector of linear/additive predictors here is $M = 21$.

Apart from the order of the linear/additive predictors, the following are (or should be) equivalent: `gaitdnbinomial()` and `negbinomial()`, `gaitdnbinomial(a.mix = 0)` and `zanegbinomial(zero = "pobs0")`, `gaitdnbinomial(i.mix = 0)` and `zinegbinomial(zero = "pstr0")`, `gaitdnbinomial(truncate = 0)` and `posnegbinomial()`. Likewise, if `a.mix` and `i.mix` are assigned a scalar then it effectively moves that scalar to `a.mlm` and `i.mlm` because there is no parameters such as `munb.i` being estimated. Thus `gaitdnbinomial(a.mix = 0)` and `gaitdnbinomial(a.mlm = 0)` are the effectively same, and ditto for `gaitdnbinomial(i.mix = 0)` and `gaitdnbinomial(i.mlm = 0)`.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, rrvglm and vgam.

The `fitted.values` slot of the fitted object, which should be extracted by the generic function `fitted`, returns the mean μ by default. See the information above on `type.fitted`.

Warning

See gaitdpoisson. Also, having `eq.ap = TRUE`, `eq.ip = TRUE` and `eq.dp = TRUE` is often needed to obtain initial values that are good enough because they borrow strength across the different operators. It is usually easy to relax these assumptions later.

This family function is under constant development and future changes will occur.

Note

If `length(a.mix)` is 1 then effectively this becomes a value of `a.mlm`. If `length(a.mix)` is 2 then an error message will be issued (overfitting really). If `length(a.mix)` is 3 then this is almost overfitting too. Hence `length(a.mix)` should be 4 or more. Ditto for `length(i.mix)` and `length(d.mix)`.

See gaitdpoisson for notes about numerical problems that can easily arise. With the NBD there is even more potential trouble that can occur. In particular, good initial values are more necessary so it pays to experiment with arguments such as `imunb.p` and `isize.p`, as well as fitting an intercept-only model first before adding covariates and using `etastart`.

Currently `max.support` is missing because only `Inf` is handled. This might change later.

Author(s)

T. W. Yee

References

Yee, T. W. and Ma, C. (2024). Generally altered, inflated, truncated and deflated regression. *Statistical Science*, **39** (in press).

See Also

Gaitdnbinom, dgaitdplot, multinomial, rootogram4, specials, plotdgaitd, spikeplot, meangaitd, KLD, gaitdpoisson, gaitdlog, gaitdzeta, multilogitlink, multinomial, goffset, Trunc, negbinomial, CommonVGAMffArguments, simulate.vlm.

Examples

```
## Not run:
i.mix <- c(5, 10, 12, 16)  # Inflate these values parametrically
i.mlm <- c(14, 15)  # Inflate these values
a.mix <- c(1, 6, 13, 20)  # Alter these values
tvec <- c(3, 11)   # Truncate these values
pstr.mlm <- 0.1  # So parallel.i = TRUE
pobs.mix <- pstr.mix <- 0.1; set.seed(1)
gdata <- data.frame(x2 = runif(nn <- 1000))
gdata <- transform(gdata, munb.p = exp(2 + 0.0 * x2),
                   size.p = exp(1))
gdata <- transform(gdata,
  y1 = rgaitdnbinom(nn, size.p, munb.p, a.mix = a.mix,
                    i.mix = i.mix,
                    pobs.mix = pobs.mix, pstr.mix = pstr.mix,
                    i.mlm = i.mlm, pstr.mlm = pstr.mlm,
                    truncate = tvec))
gaitdnbinomial(a.mix = a.mix, i.mix = i.mix, i.mlm = i.mlm)
with(gdata, table(y1))
fit1 <- vglm(y1 ~ 1, crit = "coef", trace = TRUE, data = gdata,
             gaitdnbinomial(a.mix = a.mix, i.mix = i.mix,
                            i.mlm = i.mlm,
                            parallel.i = TRUE, eq.ap = TRUE,
                            eq.ip = TRUE, truncate = tvec))
head(fitted(fit1, type.fitted = "Pstr.mix"))
head(predict(fit1))
t(coef(fit1, matrix = TRUE))  # Easier to see with t()
summary(fit1)
spikeplot(with(gdata, y1), lwd = 2)
plotdgaitd(fit1, new.plot = FALSE, offset.x = 0.2, all.lwd = 2)
## End(Not run)
```

Gaitdpois *Generally Altered, Inflated, Truncated and Deflated Poisson Distribution*

Description

Density, distribution function, quantile function and random generation for the generally altered, inflated, truncated and deflated Poisson distribution. Both parametric and nonparametric variants are supported; these are based on finite mixtures of the parent with itself and the multinomial logit model (MLM) respectively.

Usage

```
dgaitdpois(x, lambda.p, a.mix = NULL, a.mlm = NULL, i.mix = NULL,
       i.mlm = NULL, d.mix = NULL, d.mlm = NULL, truncate = NULL,
       max.support = Inf, pobs.mix = 0, pobs.mlm = 0, pstr.mix = 0,
       pstr.mlm = 0, pdip.mix = 0, pdip.mlm = 0, byrow.aid = FALSE,
       lambda.a = lambda.p, lambda.i = lambda.p,
       lambda.d = lambda.p, log = FALSE)
pgaitdpois(q, lambda.p, a.mix = NULL, a.mlm = NULL, i.mix = NULL,
       i.mlm = NULL, d.mix = NULL, d.mlm = NULL, truncate = NULL,
       max.support = Inf, pobs.mix = 0, pobs.mlm = 0, pstr.mix = 0,
       pstr.mlm = 0, pdip.mix = 0, pdip.mlm = 0, byrow.aid = FALSE,
       lambda.a = lambda.p, lambda.i = lambda.p,
       lambda.d = lambda.p, lower.tail = TRUE, checkd = FALSE)
qgaitdpois(p, lambda.p, a.mix = NULL, a.mlm = NULL, i.mix = NULL,
       i.mlm = NULL, d.mix = NULL, d.mlm = NULL, truncate = NULL,
       max.support = Inf, pobs.mix = 0, pobs.mlm = 0, pstr.mix = 0,
       pstr.mlm = 0, pdip.mix = 0, pdip.mlm = 0, byrow.aid = FALSE,
       lambda.a = lambda.p, lambda.i = lambda.p, lambda.d = lambda.p)
rgaitdpois(n, lambda.p, a.mix = NULL, a.mlm = NULL, i.mix = NULL,
       i.mlm = NULL, d.mix = NULL, d.mlm = NULL, truncate = NULL,
       max.support = Inf, pobs.mix = 0, pobs.mlm = 0, pstr.mix = 0,
       pstr.mlm = 0, pdip.mix = 0, pdip.mlm = 0, byrow.aid = FALSE,
       lambda.a = lambda.p, lambda.i = lambda.p, lambda.d = lambda.p)
```

Arguments

`x, q, p, n` Same meaning as in `Poisson`.

`log, lower.tail`
: Same meaning as in `Poisson`.

`lambda.p, lambda.a, lambda.i, lambda.d`
: Same meaning as in `Poisson`, i.e., for an ordinary Poisson distribution. The first is for the main *p*arent (or base) distribution. The next two concern the parametric variant and these distributions (usually spikes) may be *a*ltered and/or *i*nflated. The last one concerns the *d*eflated variant. Short vectors are recycled.

`truncate, max.support`
: numeric; these specify the set of truncated values. The default value of `NULL` means an empty set for the former. The latter is the maximum support value so that any value larger has been truncated (necessary because `truncate = (max.support + 1):Inf` is not allowed), hence is needed for truncating the upper tail of the distribution. Note that `max(truncate) < max.support` must be satisfied otherwise an error message will be issued.

`a.mix, i.mix, d.mix`
: Vectors of nonnegative integers; the altered, inflated and deflated values for the parametric variant. Each argument must have unique values only. Assigning argument `a.mix` means that `pobs.mix` will be used. Assigning `i.mix` means that `pstr.mix` will be used. Assigning `d.mix` means that `pdip.mix` will be used. If `a.mix` is of unit length then the default probability mass function (PMF)

evaluated at a.mix will be pobs.mix. So having a.mix = 0 corresponds to the zero-inflated Poisson distribution (see Zipois).

a.mlm, i.mlm, d.mlm
: Similar to the above, but for the nonparametric (MLM) variant. For example, assigning a.mlm means that pobs.mlm will be used. Collectively, the above 7 arguments represent 7 disjoint sets of special values and they are a proper subset of the support of the distribution.

pobs.mlm, pstr.mlm, pdip.mlm, byrow.aid
: The first three arguments are coerced into a matrix of probabilities using byrow.aid to determine the order of the elements (similar to byrow in matrix, and the .aid reinforces the behaviour that it applies to both altered, inflated and deflated cases). The first argument is recycled if necessary to become n x length(a.mlm). The second argument becomes n x length(i.mlm). The third argument becomes n x length(d.mlm). Thus these arguments are not used unless a.mlm, i.mlm and d.mlm are assigned. For deflated models, pdip.mix and pdip.mlm are positive-valued and **VGAM** will subtract these quantities; the argument deflation has been deprecated.

pobs.mix, pstr.mix, pdip.mix
: Vectors of probabilities that are recycled if necessary to length n. The first argument is used when a.mix is not NULL. The second argument is used when i.mix is not NULL. The third argument is used when d.mix is not NULL.

checkd
: Logical. If TRUE then the density is computed at floor(q) with the same parameters. This can help detect whether the PMF is invalid. If so, then NaNs are returned. See Example 2 below.

Details

These functions allow any combination of 4 operator types: truncation, alteration, inflation and deflation. The precedence is truncation, then alteration and lastly inflation and deflation. Informally, deflation can be thought of as the opposite of inflation. This order minimizes the potential interference among the operators. Loosely, a set of probabilities is set to 0 by truncation and the remaining probabilities are scaled up. Then a different set of probabilities are set to some values pobs.mix and/or pobs.mlm and the remaining probabilities are rescaled up. Then another different set of probabilities is inflated by an amount pstr.mlm and/or proportional to pstr.mix so that individual elements in this set have two sources. Then another different set of probabilities is deflated by an amount pdip.mlm and/or proportional to pdip.mix. Then all the probabilities are rescaled so that they sum to unity.

Both parametric and nonparametric variants are implemented. They usually have arguments with suffix .mix and .mlm respectively. The MLM is a loose coupling that effectively separates the *parent* (or *base*) distribution from the altered values. Values inflated nonparametrically effectively have their spikes shaved off. The .mix variant has associated with it lambda.a and lambda.i and lambda.d because it is mixture of 4 Poisson distributions with partitioned or nested support.

Any value of the support of the distribution that is altered, inflated, truncated or deflated is called a *special* value. A special value that is altered may mean that its probability increases or decreases relative to the parent distribution. An inflated special value means that its probability has increased, provided alteration elsewhere has not made it decrease in the first case. There are seven types of special values and they are represented by a.mix, a.mlm, i.mix, i.mlm, d.mix, d.mlm, truncate.

Terminology-wise, *special* values are altered or inflated or truncated or deflated, and the remaining support points that correspond directly to the parent distribution are *nonspecial* or ordinary. These functions do what `Zapois`, `Zipois`, `Pospois` collectively did plus much more.

In the notation of Yee and Ma (2023) these functions allow for the special cases: (i) GAIT–Pois(`lambda.p`)–Pois(`lambda.a`, `a.mix`, `pobs.mix`)–Pois(`lambda.i`, `i.mix`, `pstr.mix`); (ii) GAIT–Pois(`lambda.p`)–MLM(`a.mlm`, `pobs.mlm`)–MLM(`i.mlm`, `pstr.mlm`). Model (i) is totally parametric while model (ii) is the most nonparametric possible.

Value

`dgaitdpois` gives the density, `pgaitdpois` gives the distribution function, `qgaitdpois` gives the quantile function, and `rgaitdpois` generates random deviates. The default values of the arguments correspond to ordinary `dpois`, `ppois`, `qpois`, `rpois` respectively.

Warning

It is possible that the GAITD PMF is invalid because of too much inflation and/or deflation. This would result in some probabilities exceeding unity or being negative. Hence `x` should ideally contain these types of special values so that this can be detected. If so then a `NaN` is returned and a warning is issued, e.g., same as `dnorm(0, 0, sd = -1)`. To help checking, `pgaitdpois(q, ...)` calls `dgaitdpois(floor(q), ...)` if `checkd` is `TRUE`.

That is, given the parameters, it is impractical to determine whether the PMF is valid. To do this, one would have to compute the PMF at all values of its support and check that they are nonnegative and sum to unity. Hence one must be careful to input values from the parameter space, especially for inflation and deflation. See Example 2 below.

Note

Functions `Pospois` and those similar have been moved to **VGAMdata**. It is better to use `dgaitdpois(x, lambda, truncate = 0)` instead of `dposbinom(x, lambda)`, etc.

Author(s)

T. W. Yee.

References

Yee, T. W. and Ma, C. (2024). Generally altered, inflated, truncated and deflated regression. *Statistical Science*, **39** (in press).

See Also

`gaitdpoisson`, `multinomial`, `specials`, `spikeplot`, `dgaitdplot`, `Zapois`, `Zipois`, `Pospois` `Poisson`; `Gaitdbinom`, `Gaitdnbinom`, `Gaitdlog`, `Gaitdzeta`.

Examples

```
 # Example 1
ivec <- c(6, 14); avec <- c(8, 11); lambda <- 10; xgrid <- 0:25
tvec <- 15; max.support <- 20; pobs.mix <- 0.05; pstr.i <- 0.25
dvec <- 13; pdip.mlm <- 0.05; pobs.mlm <- 0.05
(ddd <- dgaitdpois(xgrid, lambda, lambda.a = lambda + 5,
   truncate = tvec, max.support = max.support, pobs.mix = pobs.mix,
   pobs.mlm = pobs.mlm, a.mlm = avec,
   pdip.mlm = pdip.mlm, d.mlm = dvec,
   pstr.mix = pstr.i, i.mix = ivec))
## Not run:  dgaitdplot(lambda, ylab = "Probability", xlab = "x",
   truncate = tvec, max.support = max.support, pobs.mix = pobs.mix,
   pobs.mlm = pobs.mlm, a.mlm = avec, all.lwd = 3,
   pdip.mlm = pdip.mlm, d.mlm = dvec,
   pstr.mix = pstr.i, i.mix = ivec, deflation = TRUE,
   main = "GAITD Combo PMF---Poisson Parent")
## End(Not run)

# Example 2: detection of an invalid PMF
xgrid <- 1:3  # Does not cover the special values purposely
(ddd <- dgaitdpois(xgrid, 1, pdip.mlm = 0.1, d.mlm = 5,
                  pstr.mix = 0.95, i.mix = 0))  # Undetected
xgrid <- 0:13  # Wider range so this detects the problem
(ddd <- dgaitdpois(xgrid, 1, pdip.mlm = 0.1, d.mlm = 5,
                   pstr.mix = 0.95, i.mix = 0))  # Detected
sum(ddd, na.rm = TRUE)  # Something gone awry
```

`gaitdpoisson` *Generally Altered, Inflated, Truncated and Deflated Poisson Regression*

Description

Fits a generally altered, inflated, truncated and deflated Poisson regression by MLE. The GAITD combo model having 7 types of special values is implemented. This allows mixtures of Poissons on nested and/or partitioned support as well as a multinomial logit model for (nonparametric) altered, inflated and deflated values. Truncation may include the upper tail.

Usage

```
gaitdpoisson(a.mix = NULL, i.mix = NULL, d.mix = NULL,
     a.mlm = NULL, i.mlm = NULL, d.mlm = NULL,
     truncate = NULL, max.support = Inf,
     zero = c("pobs", "pstr", "pdip"),
     eq.ap = TRUE, eq.ip = TRUE, eq.dp = TRUE,
     parallel.a = FALSE, parallel.i = FALSE, parallel.d = FALSE,
     llambda.p = "loglink", llambda.a = llambda.p,
     llambda.i = llambda.p, llambda.d = llambda.p,
     type.fitted = c("mean", "lambdas", "pobs.mlm", "pstr.mlm",
```

```
"pdip.mlm", "pobs.mix", "pstr.mix", "pdip.mix",
"Pobs.mix", "Pstr.mix", "Pdip.mix", "nonspecial",
"Numer", "Denom.p", "sum.mlm.i", "sum.mix.i",
"sum.mlm.d", "sum.mix.d", "ptrunc.p",
"cdf.max.s"), gpstr.mix = ppoints(7) / 3,
gpstr.mlm = ppoints(7) / (3 + length(i.mlm)),
imethod = 1, mux.init = c(0.75, 0.5, 0.75),
ilambda.p = NULL, ilambda.a = ilambda.p,
ilambda.i = ilambda.p, ilambda.d = ilambda.p,
ipobs.mix = NULL, ipstr.mix = NULL, ipdip.mix = NULL,
ipobs.mlm = NULL, ipstr.mlm = NULL, ipdip.mlm = NULL,
byrow.aid = FALSE, ishrinkage = 0.95, probs.y = 0.35)
```

Arguments

`truncate, max.support`
: Vector of truncated values, i.e., nonnegative integers. For the first seven arguments (for the *special* values) a `NULL` stands for an empty set, and the seven sets must be mutually disjoint. Argument `max.support` enables RHS-truncation, i.e., something equivalent to `truncate = (U+1):Inf` for some upper support point `U` specified by `max.support`.

`a.mix, i.mix, d.mix`
: Vector of altered and inflated values corresponding to finite mixture models. These are described as *parametric* or structured.

The parameter `lambda.p` is always estimated. If `length(a.mix)` is 1 or more then the parameter `pobs.mix` is estimated. If `length(i.mix)` is 1 or more then the parameter `pstr.mix` is estimated. If `length(d.mix)` is 1 or more then the parameter `pdip.mix` is estimated.

If `length(a.mix)` is 2 or more then the parameter `lambda.a` is estimated. If `length(i.mix)` is 2 or more then the parameter `lambda.i` is estimated. If `length(d.mix)` is 2 or more then the parameter `lambda.d` is estimated.

If `length(a.mix) == 1`, `length(i.mix) == 1` or `length(d.mix) == 1` then `lambda.a`, `lambda.i` and `lambda.d` are unidentifiable and therefore ignored. In such cases it would be equivalent to moving `a.mix` into `a.mlm`, etc.

Due to its great flexibility, it is easy to misuse this function and ideally the values of the above arguments should be well justified by the application on hand. Adding inappropriate or unnecessary values to these arguments willy-nilly is a recipe for disaster, especially for `i.mix` and `d.mix`. Using `a.mlm` effectively removes a subset of the data from the main analysis, therefore may result in a substantial loss of efficiency. For seeped values, `a.mix`, `a.mlm`, `d.mix` and `d.mlm` can be used only. Heaped values can be handled by `i.mlm` and `i.mix`, as well as `a.mix` and `a.mlm`. Because of the NBP reason below, it sometimes may be necessary to specify deflated values to altered values.

`a.mlm, i.mlm, d.mlm`
: Vector of altered, inflated and deflated values corresponding to the multinomial logit model (MLM) probabilities of observing those values—see `multinomial`. These are described as *nonparametric* or unstructured.

`llambda.p, llambda.a, llambda.i, llambda.d`
: Link functions for the parent, altered, inflated and deflated distributions respectively. See `Links` for more choices and information.

`eq.ap, eq.ip, eq.dp`
: Single logical each. Constrain the rate parameters to be equal? See `CommonVGAMffArguments` for information. Having all three arguments `TRUE` gives greater stability in the estimation because of fewer parameters and therefore fewer initial values needed, however if so then one should try relax some of the arguments later.

  For the GIT–Pois submodel, after plotting the responses, if the distribution of the spikes above the nominal probabilities has roughly the same shape as the ordinary values then setting `eq.ip = TRUE` would be a good idea so that `lambda.i == lambda.p`. And if `i.mix` is of length 2 or a bit more, then `TRUE` should definitely be entertained. Likewise, for heaped or seeped data, setting `eq.ap = TRUE` (so that `lambda.p == lambda.p`) would be a good idea for the GAT–Pois if the shape of the altered probabilities is roughly the same as the parent distribution.

`parallel.a, parallel.i, parallel.d`
: Single logical each. Constrain the MLM probabilities to be equal? If so then this applies to all `length(a.mlm)` `pobs.mlm` probabilities or all `length(i.mlm)` `pstr.mlm` probabilities or all `length(d.mlm)` `pdip.mlm` probabilities. See `CommonVGAMffArguments` for information. The default means that the probabilities are generally unconstrained and unstructured and will follow the shape of the data. See `constraints`.

`type.fitted`
: See `CommonVGAMffArguments` and below for information. The first value is the default, and this is usually the unconditional mean. Choosing an irrelevant value may result in an `NA` being returned and a warning, e.g., `"pstr.mlm"` for a nonparametric GAT model.

  The choice `"lambdas"` returns a matrix with at least one column and up to three others, corresponding to all those estimated. In order, their `colnames` are `"lambda.p"`, `"lambda.a"`, `"lambda.i"` and `"lambda.d"`. For other distributions such as `gaitdlog` `type.fitted = "shapes"` is permitted and the `colnames` are `"shape.p"`, `"shape.a"`, `"shape.i"` and `"shape.d"`, etc.

  Option `"Pobs.mix"` provides more detail about `"pobs.mix"` by returning a matrix whose columns correspond to each altered value; the row sums (`rowSums`) of this matrix is `"pobs.mix"`. Likewise `"Pstr.mix"` about `"pstr.mix"` and `"Pdip.mix"` about `"pdip.mix"`.

  The choice `"cdf.max.s"` is the CDF evaluated at `max.support` using the parent distribution, e.g., `ppois(max.support, lambda.p)` for `gaitdpoisson`. The value should be 1 if `max.support = Inf` (the default). The choice `"nonspecial"` is the probability of a nonspecial value. The choices `"Denom.p"` and `"Numer"` are quantities found in the GAITD combo PMF and are for convenience only.

  The choice `type.fitted = "pobs.mlm"` returns a matrix whose columns are the altered probabilities (Greek symbol ω_s). The choice `"pstr.mlm"` returns a matrix whose columns are the inflated probabilities (Greek symbol ϕ_s). The choice `"pdip.mlm"` returns a matrix whose columns are the deflated probabilities (Greek symbol ψ_s).

  The choice `"ptrunc.p"` returns the probability of having a truncated value with respect to the parent distribution. It includes any truncated values in the upper tail beyond `max.support`. The probability of a value less than or equal to `max.support` with respect to the parent distribution is `"cdf.max.s"`.

The choice "sum.mlm.i" adds two terms. This gives the probability of an inflated value, and the formula can be loosely written down as something like "pstr.mlm" + "Numer" * dpois(i.mlm, lambda.p) / "Denom.p". The other three "sum.m*" arguments are similar.

gpstr.mix, gpstr.mlm
: See CommonVGAMffArguments for information. Gridsearch values for the two parameters. If failure occurs try a finer grid, especially closer to 0, and/or experiment with mux.init.

imethod, ipobs.mix, ipstr.mix, ipdip.mix
: See CommonVGAMffArguments for information. Good initial values are difficult to compute because of the great flexibility of GAITD regression, therefore it is often necessary to use these arguments. A careful examination of a spikeplot of the data should lead to good choices.

ipobs.mlm, ipstr.mlm, ipdip.mlm
: See CommonVGAMffArguments for information.

mux.init
: Numeric, of length 3. General downward multiplier for initial values for the sample proportions (MLEs actually). This is under development and more details are forthcoming. In general, 1 means unchanged and values should lie in (0, 1], and values about 0.5 are recommended. The elements apply in order to altered, inflated and deflated (no distinction between mix and MLM).

ilambda.p, ilambda.a, ilambda.i, ilambda.d
: Initial values for the rate parameters; see CommonVGAMffArguments for information.

probs.y, ishrinkage
: See CommonVGAMffArguments for information.

byrow.aid
: Details are at Gaitdpois.

zero
: See CommonVGAMffArguments for information. By default, all the MLM probabilities are modelled as simple as possible (intercept-only) to help avoid numerical problems, especially when there are many covariates. The Poisson means are modelled by the covariates, and the default zero vector is pruned of any irrelevant values. To model all the MLM probabilities with covariates set zero = NULL, however, the number of regression coefficients could be excessive.

  For the MLM probabilities, to model pobs.mix only with covariates set zero = c('pstr', 'pobs.mlm', 'pdip'). Likewise, to model pstr.mix only with covariates set zero = c('pobs', 'pstr.mlm', 'pdip').

  It is noted that, amongst other things, zipoisson and zipoissonff differ with respect to zero, and ditto for zapoisson and zapoissonff.

Details

The full GAITD–Pois combo model may be fitted with this family function. There are seven types of special values and all arguments for these may be used in a single model. Here, the MLM represents the nonparametric while the Pois refers to the Poisson mixtures. The defaults for this function correspond to an ordinary Poisson regression so that poissonff is called instead. A MLM with only one probability to model is equivalent to logistic regression (binomialff and logitlink).

The order of the linear/additive predictors is best explained by an example. Suppose a combo model has length(a.mix) > 2 and length(i.mix) > 2, length(d.mix) > 2, a.mlm = 3:5, i.mlm = 6:9

and d.mlm = 10:12, say. Then loglink(lambda.p) is the first. The second is multilogitlink(pobs.mix) followed by loglink(lambda.a) because a.mix is long enough. The fourth is multilogitlink(pstr.mix) followed by loglink(lambda.i) because i.mix is long enough. The sixth is multilogitlink(pdip.mix) followed by loglink(lambda.d) because d.mix is long enough. Next are the probabilities for the a.mlm values. Then are the probabilities for the i.mlm values. Lastly are the probabilities for the d.mlm values. All the probabilities are estimated by one big MLM and effectively the "(Others)" column of left over probabilities is associated with the nonspecial values. These might be called the *nonspecial baseline probabilities* (NBP). The dimension of the vector of linear/additive predictors here is $M = 17$.

Two mixture submodels that may be fitted can be abbreviated GAT–Pois or GIT–Pois. For the GAT model the distribution being fitted is a (spliced) mixture of two Poissons with differing (partitioned) support. Likewise, for the GIT model the distribution being fitted is a mixture of two Poissons with nested support. The two rate parameters may be constrained to be equal using eq.ap and eq.ip.

A good first step is to apply spikeplot for selecting candidate values for altering, inflating and deflating. Deciding between parametrically or nonparametrically can also be determined from examining the spike plot. Misspecified a.mix/a.mlm/i.mix/i.mlm/d.mix/d.mlm will result in convergence problems (setting trace = TRUE is a *very* good idea.) This function currently does not handle multiple responses. Further details are at Gaitdpois.

A well-conditioned data–model combination should pose no difficulties for the automatic starting value selection being successful. Failure to obtain initial values from this self-starting family function indicates the degree of inflation/deflation may be marginal and/or a misspecified model. If this problem is worth surmounting the arguments to focus on especially are mux.init, gpstr.mix, gpstr.mlm, ipdip.mix and ipdip.mlm. See below for the stepping-stone trick.

Apart from the order of the linear/additive predictors, the following are (or should be) equivalent: gaitdpoisson() and poissonff(), gaitdpoisson(a.mix = 0) and zapoisson(zero = "pobs0"), gaitdpoisson(i.mix = 0) and zipoisson(zero = "pstr0"), gaitdpoisson(truncate = 0) and pospoisson(). Likewise, if a.mix and i.mix are assigned a scalar then it effectively moves that scalar to a.mlm and i.mlm because there is no lambda.a or lambda.i being estimated. Thus gaitdpoisson(a.mix = 0) and gaitdpoisson(a.mlm = 0) are the effectively same, and ditto for gaitdpoisson(i.mix = 0) and gaitdpoisson(i.mlm = 0).

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, rrvglm and vgam.

The fitted.values slot of the fitted object, which should be extracted by the generic function fitted, returns the mean μ by default. See the information above on type.fitted.

Warning

Amateurs tend to be overzealous fitting zero-inflated models when the fitted mean is low—the warning of ziP should be heeded. For GAITD regression the warning applies more strongly and generally; here to *all* i.mix, i.mlm, d.mix and d.mlm values, not just 0. Even one misspecified special value usually will cause convergence problems.

Default values for this and similar family functions may change in the future, e.g., eq.ap and eq.ip. Important internal changes might occur too, such as the ordering of the linear/additive predictors and the quantities returned as the fitted values.

Using `i.mlm` requires more caution than `a.mlm` because gross inflation is ideally needed for it to work safely. Ditto for `i.mix` versus `a.mix`. Data exhibiting deflation or little to no inflation will produce numerical problems, hence set `trace = TRUE` to monitor convergence. More than c.10 IRLS iterations should raise suspicion.

Ranking the four operators by difficulty, the easiest is truncation followed by alteration, then inflation and the most difficult is deflation. The latter needs good initial values and the current default will probably not work on some data sets. Studying the spikeplot is time very well spent. In general it is very easy to specify an *overfitting* model so it is a good idea to split the data into training and test sets.

This function is quite memory-hungry with respect to `length(c(a.mix, i.mix, d.mix, a.mlm, i.mlm, d.mlm))`. On consuming something different, because all values of the NBP vector need to be positive it pays to be economical with respect to `d.mlm` especially so that one does not consume up probabilities unnecessarily so to speak.

It is often a good idea to set `eq.ip = TRUE`, especially when `length(i.mix)` is not much more than 2 or the values of `i.mix` are not spread over the range of the response. This way the estimation can borrow strength from both the inflated and non-inflated values. If the `i.mix` values form a single small cluster then this can easily create estimation difficulties—the idea is somewhat similar to multicollinearity. The same holds for `d.mix`.

Note

Numerical problems can easily arise because of the exceeding flexibility of this distribution and/or the lack of sizeable inflation/deflation; it is a good idea to gain experience with simulated data first before applying it to real data. Numerical problems may arise if any of the special values are in remote places of the support, e.g., a value `y` such that `dpois(y, lambda.p)` is very close to 0. This is because the ratio of two tiny values can be unstable.

Good initial values may be difficult to obtain using self-starting procedures, especially when there are covariates. If so, then it is advisable to use a trick: fit an intercept-only model first and then use `etastart = predict(int.only.model)` to fit the model with covariates. This uses the simpler model as a stepping-stone.

The labelling of the linear/additive predictors has been abbreviated to reduce space. For example, `multilogitlink(pobs.mix)` and `multilogitlink(pstr.mix)` would be more accurately `multilogitlink(cbind(pobs.mix, pstr.mix))` because one grand MLM is fitted. This shortening may result in modifications needed in other parts of **VGAM** to compensate.

Because estimation involves a MLM, the restricted parameter space means that if the dip probabilities are large then the NBP may become too close to 0. If this is so then there are tricks to avoid a negative NBP. One of them is to model as many values of `d.mlm` as `d.mix`, hence the dip probabilities become modelled via the deflation distribution instead. Another trick to alter those special values rather than deflating them if the dip probabilities are large.

Due to its complexity, the HDE test `hdeff` is currently unavailable for GAITD regressions.

Randomized quantile residuals (RQRs) are available; see `residualsvglm`.

Author(s)

T. W. Yee

References

Yee, T. W. and Ma, C. (2024). Generally altered, inflated, truncated and deflated regression. *Statistical Science*, **39** (in press).

See Also

Gaitdpois, multinomial, rootogram4, specials, plotdgaitd, spikeplot, meangaitd, KLD, goffset, Trunc, gaitdnbinomial, gaitdlog, gaitdzeta, multilogitlink, multinomial, residualsvglm, poissonff, zapoisson, zipoisson, pospoisson, CommonVGAMffArguments, simulate.vlm.

Examples

```
i.mix <- c(5, 10)  # Inflate these values parametrically
i.mlm <- c(14, 15)  # Inflate these values
a.mix <- c(1, 13)  # Alter these values
tvec <- c(3, 11)   # Truncate these values
pstr.mlm <- 0.1  # So parallel.i = TRUE
pobs.mix <- pstr.mix <- 0.1
max.support <- 20; set.seed(1)
gdata <- data.frame(x2 = runif(nn <- 1000))
gdata <- transform(gdata, lambda.p = exp(2 + 0.0 * x2))
gdata <- transform(gdata,
  y1 = rgaitdpois(nn, lambda.p, a.mix = a.mix, i.mix = i.mix,
                  pobs.mix = pobs.mix, pstr.mix = pstr.mix,
                  i.mlm = i.mlm, pstr.mlm = pstr.mlm,
                  truncate = tvec, max.support = max.support))
gaitdpoisson(a.mix = a.mix, i.mix = i.mix, i.mlm = i.mlm)
with(gdata, table(y1))
fit1 <- vglm(y1 ~ 1, crit = "coef", trace = TRUE, data = gdata,
             gaitdpoisson(a.mix = a.mix, i.mix = i.mix,
                          i.mlm = i.mlm, parallel.i = TRUE,
                          eq.ap = TRUE, eq.ip = TRUE, truncate =
                          tvec, max.support = max.support))
head(fitted(fit1, type.fitted = "Pstr.mix"))
head(predict(fit1))
t(coef(fit1, matrix = TRUE))  # Easier to see with t()
summary(fit1)  # No HDE test by default but HDEtest = TRUE is ideal
## Not run:  spikeplot(with(gdata, y1), lwd = 2)
plotdgaitd(fit1, new.plot = FALSE, offset.x = 0.2, all.lwd = 2)
## End(Not run)
```

Gaitdzeta   *Generally Altered, Inflated and Truncated and Deflated Zeta Distribution*

Description

Density, distribution function, quantile function and random generation for the generally altered, inflated, truncated and deflated zeta distribution. Both parametric and nonparametric variants are supported; these are based on finite mixtures of the parent with itself and the multinomial logit model (MLM) respectively.

Usage

```
dgaitdzeta(x, shape.p, a.mix = NULL, a.mlm = NULL,
           i.mix = NULL, i.mlm = NULL,
           d.mix = NULL, d.mlm = NULL, truncate = NULL,
           max.support = Inf, pobs.mix = 0, pobs.mlm = 0,
           pstr.mix = 0, pstr.mlm = 0,
           pdip.mix = 0, pdip.mlm = 0,
           byrow.aid = FALSE,
           shape.a = shape.p, shape.i = shape.p, shape.d = shape.p,
           log = FALSE)
pgaitdzeta(q, shape.p, a.mix = NULL, a.mlm = NULL,
           i.mix = NULL, i.mlm = NULL,
           d.mix = NULL, d.mlm = NULL, truncate = NULL,
           max.support = Inf, pobs.mix = 0, pobs.mlm = 0,
           pstr.mix = 0, pstr.mlm = 0,
           pdip.mix = 0, pdip.mlm = 0,
           byrow.aid = FALSE,
           shape.a = shape.p, shape.i = shape.p, shape.d = shape.p,
           lower.tail = TRUE)
qgaitdzeta(p, shape.p, a.mix = NULL, a.mlm = NULL,
           i.mix = NULL, i.mlm = NULL,
           d.mix = NULL, d.mlm = NULL, truncate = NULL,
           max.support = Inf, pobs.mix = 0, pobs.mlm = 0,
           pstr.mix = 0, pstr.mlm = 0,
           pdip.mix = 0, pdip.mlm = 0,
           byrow.aid = FALSE,
           shape.a = shape.p, shape.i = shape.p, shape.d = shape.p)
rgaitdzeta(n, shape.p, a.mix = NULL, a.mlm = NULL,
           i.mix = NULL, i.mlm = NULL,
           d.mix = NULL, d.mlm = NULL, truncate = NULL,
           max.support = Inf, pobs.mix = 0, pobs.mlm = 0,
           pstr.mix = 0, pstr.mlm = 0,
           pdip.mix = 0, pdip.mlm = 0,
           byrow.aid = FALSE,
           shape.a = shape.p, shape.i = shape.p, shape.d = shape.p)
```

Arguments

`x, q, p, n, log, lower.tail`
: Same meaning as in `dzeta`.

`shape.p, shape.a, shape.i, shape.d`
: Same meaning as shape for `dzeta`, i.e., for an ordinary zeta distribution. See `Gaitdpois` for generic information.

`truncate, max.support`
: See `Gaitdpois` for generic information.

a.mix, i.mix, d.mix
See Gaitdpois for generic information.

a.mlm, i.mlm, d.mlm
See Gaitdpois for generic information.

pobs.mlm, pstr.mlm, pdip.mlm, byrow.aid
See Gaitdpois for generic information.

pobs.mix, pstr.mix, pdip.mix
See Gaitdpois for generic information.

Details

These functions for the zeta distribution are analogous to the Poisson, hence most details have been put in Gaitdpois. These functions do what Oazeta, Oizeta, Otzeta collectively did plus much more.

Value

dgaitdzeta gives the density, pgaitdzeta gives the distribution function, qgaitdzeta gives the quantile function, and rgaitdzeta generates random deviates. The default values of the arguments correspond to ordinary dzeta, pzeta, qzeta, rzeta respectively.

Warning

See Gaitdpois about the dangers of too much inflation and/or deflation on GAITD PMFs, and the difficulties detecting such.

Note

See Gaitdpois for general information also relevant to this parent distribution.

Author(s)

T. W. Yee.

See Also

gaitdzeta, Gaitdpois, dgaitdplot, multinomial, Oazeta, Oizeta, Otzeta.

Examples

```
ivec <- c(2, 10); avec <- ivec + 4; shape <- 0.95; xgrid <- 0:29
tvec <- 15; max.support <- 25; pobs.a <- 0.10; pstr.i <- 0.15
(ddd <- dgaitdzeta(xgrid, shape, truncate = tvec,
   max.support = max.support, pobs.mix = pobs.a,
   a.mix = avec, pstr.mix = pstr.i, i.mix = ivec))
## Not run: plot(xgrid, ddd, type = "n", ylab = "Probability",
              xlab = "x", main = "GAIT PMF---Zeta Parent")
mylwd <- 0.5
abline(v = avec, col = 'blue', lwd = mylwd)
abline(v = ivec, col = 'purple', lwd = mylwd)
```

```
abline(v = tvec, col = 'tan', lwd = mylwd)
abline(v = max.support, col = 'magenta', lwd = mylwd)
abline(h = c(pobs.a, pstr.i, 0:1), col = 'gray', lty = "dashed")
lines(xgrid, dzeta(xgrid, shape), col='gray', lty="dashed")  # f_{\pi}
lines(xgrid, ddd, type = "h", col = "pink", lwd = 3)  # GAIT PMF
points(xgrid[ddd == 0], ddd[ddd == 0], pch = 16, col = 'tan', cex = 2)

## End(Not run)
```

---

gaitdzeta *Generally Altered, Inflated, Truncated and Deflated Zeta Regression*

Description

Fits a generally altered, inflated, truncated and deflated zeta regression by MLE. The GAITD combo model having 7 types of special values is implemented. This allows mixtures of zetas on nested and/or partitioned support as well as a multinomial logit model for altered, inflated and deflated values.

Usage

```
gaitdzeta(a.mix = NULL, i.mix = NULL, d.mix = NULL,
         a.mlm = NULL, i.mlm = NULL, d.mlm = NULL,
         truncate = NULL, max.support = Inf,
         zero = c("pobs", "pstr", "pdip"), eq.ap = TRUE, eq.ip = TRUE,
         eq.dp = TRUE, parallel.a = FALSE,
         parallel.i = FALSE, parallel.d = FALSE,
         lshape.p = "loglink", lshape.a = lshape.p,
         lshape.i = lshape.p, lshape.d = lshape.p,
         type.fitted = c("mean", "shapes", "pobs.mlm", "pstr.mlm",
         "pdip.mlm", "pobs.mix", "pstr.mix", "pdip.mix", "Pobs.mix",
         "Pstr.mix", "Pdip.mix", "nonspecial",
         "Numer", "Denom.p", "sum.mlm.i", "sum.mix.i", "sum.mlm.d",
         "sum.mix.d", "ptrunc.p", "cdf.max.s"),
         gshape.p = -expm1(-ppoints(7)), gpstr.mix = ppoints(7) / 3,
         gpstr.mlm = ppoints(7) / (3 + length(i.mlm)),
         imethod = 1, mux.init = c(0.75, 0.5, 0.75),
         ishape.p = NULL, ishape.a = ishape.p,
         ishape.i = ishape.p, ishape.d = ishape.p,
         ipobs.mix = NULL, ipstr.mix = NULL, ipdip.mix = NULL,
         ipobs.mlm = NULL, ipstr.mlm = NULL, ipdip.mlm = NULL,
         byrow.aid = FALSE, ishrinkage = 0.95, probs.y = 0.35)
```

Arguments

`truncate, max.support`
: See gaitdpoisson. Only `max.support = Inf` is allowed because some equations are intractable.

`a.mix, i.mix, d.mix`
: See `gaitdpoisson`.

`a.mlm, i.mlm, d.mlm`
: See `gaitdpoisson`.

`lshape.p, lshape.a, lshape.i, lshape.d`
: Link functions. See `gaitdpoisson` and `Links` for more choices and information. Actually, it is usually a good idea to set these arguments equal to `zetaffMlink` because the log-mean is the first linear/additive predictor so it is like a Poisson regression.

`eq.ap, eq.ip, eq.dp`
: Single logical each. See `gaitdpoisson`

`parallel.a, parallel.i, parallel.d`
: Single logical each. See `gaitdpoisson`.

`type.fitted, mux.init`
: See `gaitdpoisson`.

`imethod, ipobs.mix, ipstr.mix, ipdip.mix`
: See `CommonVGAMffArguments` and `gaitdpoisson` for information.

`ipobs.mlm, ipstr.mlm, ipdip.mlm, byrow.aid`
: See `CommonVGAMffArguments` and `gaitdpoisson` for information.

`gpstr.mix, gpstr.mlm`
: See `CommonVGAMffArguments` and `gaitdpoisson` for information.

`gshape.p, ishape.p`
: See `CommonVGAMffArguments` and `gaitdpoisson` for information. The former is used only if the latter is not given. Practical experience has shown that good initial values are needed, so if convergence is not obtained then try a finer grid.

`ishape.a, ishape.i, ishape.d`
: See `CommonVGAMffArguments` and `gaitdpoisson` for information.

`probs.y, ishrinkage`
: See `CommonVGAMffArguments` for information.

`zero`
: See `gaitdpoisson` and `CommonVGAMffArguments` for information.

Details

Many details to this family function can be found in `gaitdpoisson` because it is also a 1-parameter discrete distribution. This function currently does not handle multiple responses. Further details are at `Gaitdzeta`.

As alluded to above, when there are covariates it is much more interpretable to model the mean rather than the shape parameter. Hence `zetaffMlink` is recommended. (This might become the default in the future.) So installing **VGAMextra** is a good idea.

Apart from the order of the linear/additive predictors, the following are (or should be) equivalent: `gaitdzeta()` and `zetaff()`, `gaitdzeta(a.mix = 1)` and `oazeta(zero = "pobs1")`, `gaitdzeta(i.mix = 1)` and `oizeta(zero = "pstr1")`, `gaitdzeta(truncate = 1)` and `otzeta()`. The functions `oazeta`, `oizeta` and `otzeta` have been placed in **VGAMdata**.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, `rrvglm` and `vgam`.

Warning

See gaitdpoisson.

Note

See gaitdpoisson.

Author(s)

T. W. Yee

See Also

Gaitdzeta, zetaff, zetaffMlink, Gaitdpois, gaitdpoisson, gaitdlog, spikeplot, goffset, Trunc, oazeta, oizeta, otzeta, CommonVGAMffArguments, rootogram4, simulate.vlm.

Examples

```
## Not run:
avec <- c(5, 10)  # Alter these values parametrically
ivec <- c(3, 15)  # Inflate these values
tvec <- c(6, 7)   # Truncate these values
set.seed(1); pobs.a <- pstr.i <- 0.1
gdata <- data.frame(x2 = runif(nn <- 1000))
gdata <- transform(gdata, shape.p = logitlink(2, inverse = TRUE))
gdata <- transform(gdata,
  y1 = rgaitdzeta(nn, shape.p, a.mix = avec, pobs.mix = pobs.a,
                  i.mix = ivec, pstr.mix = pstr.i, truncate = tvec))
gaitdzeta(a.mix = avec, i.mix = ivec)
with(gdata, table(y1))
spikeplot(with(gdata, y1), las = 1)
fit7 <- vglm(y1 ~ 1, trace = TRUE, data = gdata, crit = "coef",
             gaitdzeta(i.mix = ivec, truncate = tvec,
                       a.mix = avec, eq.ap = TRUE, eq.ip = TRUE))
head(fitted(fit7, type.fitted = "Pstr.mix"))
head(predict(fit7))
t(coef(fit7, matrix = TRUE))  # Easier to see with t()
summary(fit7)
spikeplot(with(gdata, y1), lwd = 2, ylim = c(0, 0.6), xlim = c(0, 20))
plotdgaitd(fit7, new.plot = FALSE, offset.x = 0.2, all.lwd = 2)

## End(Not run)
```

---

gamma1 *1-parameter Gamma Regression Family Function*

Description

Estimates the 1-parameter gamma distribution by maximum likelihood estimation.

Usage

```
gamma1(link = "loglink", zero = NULL, parallel = FALSE,
       type.fitted = c("mean", "percentiles", "Qlink"),
       percentiles = 50)
```

Arguments

`link` Link function applied to the (positive) *shape* parameter. See `Links` for more choices and general information.

`zero, parallel` Details at `CommonVGAMffArguments`.

`type.fitted, percentiles`
See `CommonVGAMffArguments` for information. Using `"Qlink"` is for quantile-links in **VGAMextra**.

Details

The density function is given by

$$f(y) = \exp(-y) \times y^{shape-1}/\Gamma(shape)$$

for $shape > 0$ and $y > 0$. Here, $\Gamma(shape)$ is the gamma function, as in `gamma`. The mean of Y (returned as the default fitted values) is $\mu = shape$, and the variance is $\sigma^2 = shape$.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Note

This **VGAM** family function can handle a multiple responses, which is inputted as a matrix.

The parameter $shape$ matches with `shape` in `rgamma`. The argument `rate` in `rgamma` is assumed 1 for this family function, so that `scale = 1` is used for calls to `dgamma`, `qgamma`, etc.

If $rate$ is unknown use the family function `gammaR` to estimate it too.

Author(s)

T. W. Yee

References

Most standard texts on statistical distributions describe the 1-parameter gamma distribution, e.g.,

Forbes, C., Evans, M., Hastings, N. and Peacock, B. (2011). *Statistical Distributions*, Hoboken, NJ, USA: John Wiley and Sons, Fourth edition.

See Also

`gammaR` for the 2-parameter gamma distribution, `lgamma1`, `lindley`, `simulate.vlm`, `gammaff.mm`.

Examples

```
gdata <- data.frame(y = rgamma(n = 100, shape = exp(3)))
fit <- vglm(y ~ 1, gamma1, data = gdata, trace = TRUE, crit = "coef")
coef(fit, matrix = TRUE)
Coef(fit)
summary(fit)
```

---

gamma2 *2-parameter Gamma Regression Family Function*

Description

Estimates the 2-parameter gamma distribution by maximum likelihood estimation.

Usage

```
gamma2(lmu = "loglink", lshape = "loglink", imethod = 1,  ishape = NULL,
       parallel = FALSE, deviance.arg = FALSE, zero = "shape")
```

Arguments

`lmu, lshape`	Link functions applied to the (positive) *mu* and *shape* parameters (called μ and a respectively). See `Links` for more choices.
`ishape`	Optional initial value for *shape*. A `NULL` means a value is computed internally. If a failure to converge occurs, try using this argument. This argument is ignored if used within `cqo`; see the `iShape` argument of `qrrvglm.control` instead.
`imethod`	An integer with value `1` or `2` which specifies the initialization method for the μ parameter. If failure to converge occurs try another value (and/or specify a value for `ishape`).
`deviance.arg`	Logical. If `TRUE`, the deviance function is attached to the object. Under ordinary circumstances, it should be left alone because it really assumes the shape parameter is at the maximum likelihood estimate. Consequently, one cannot use that criterion to minimize within the IRLS algorithm. It should be set `TRUE` only when used with `cqo` under the fast algorithm.
`zero`	See `CommonVGAMffArguments` for information.
`parallel`	Details at `CommonVGAMffArguments`. If `parallel = TRUE` then the constraint is not applied to the intercept.

Details

This distribution can model continuous skewed responses. The density function is given by

$$f(y; \mu, a) = \frac{\exp(-ay/\mu) \times (ay/\mu)^{a-1} \times a}{\mu \times \Gamma(a)}$$

for $\mu > 0$, $a > 0$ and $y > 0$. Here, $\Gamma(\cdot)$ is the gamma function, as in `gamma`. The mean of Y is $\mu = \mu$ (returned as the fitted values) with variance $\sigma^2 = \mu^2/a$. If $0 < a < 1$ then the density has a pole at the origin and decreases monotonically as y increases. If $a = 1$ then this corresponds to the exponential distribution. If $a > 1$ then the density is zero at the origin and is unimodal with mode at $y = \mu - \mu/a$; this can be achieved with `lshape="logloglink"`.

By default, the two linear/additive predictors are $\eta_1 = \log(\mu)$ and $\eta_2 = \log(a)$. This family function implements Fisher scoring and the working weight matrices are diagonal.

This **VGAM** family function handles *multivariate* responses, so that a matrix can be used as the response. The number of columns is the number of species, say, and `zero=-2` means that *all* species have a shape parameter equalling a (different) intercept only.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Note

The response must be strictly positive. A moment estimator for the shape parameter may be implemented in the future.

If `mu` and `shape` are vectors, then `rgamma(n = n, shape = shape, scale = mu/shape)` will generate random gamma variates of this parameterization, etc.; see `GammaDist`.

Author(s)

T. W. Yee

References

The parameterization of this **VGAM** family function is the 2-parameter gamma distribution described in the monograph

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*, 2nd ed. London: Chapman & Hall.

See Also

`gamma1` for the 1-parameter gamma distribution, `gammaR` for another parameterization of the 2-parameter gamma distribution that is directly matched with `rgamma`, `bigamma.mckay` for *a* bivariate gamma distribution, `gammaff.mm` for another, `expexpff`, `GammaDist`, `CommonVGAMffArguments`, `simulate.vlm`, `negloglink`.

Examples

```
# Essentially a 1-parameter gamma
gdata <- data.frame(y = rgamma(n = 100, shape = exp(1)))
fit1 <- vglm(y ~ 1, gamma1, data = gdata)
fit2 <- vglm(y ~ 1, gamma2, data = gdata, trace = TRUE, crit = "coef")
coef(fit2, matrix = TRUE)
c(Coef(fit2), colMeans(gdata))

# Essentially a 2-parameter gamma
```

```
gdata <- data.frame(y = rgamma(n = 500, rate = exp(-1), shape = exp(2)))
fit2 <- vglm(y ~ 1, gamma2, data = gdata, trace = TRUE, crit = "coef")
coef(fit2, matrix = TRUE)
c(Coef(fit2), colMeans(gdata))
summary(fit2)
```

---

gammaff.mm *Multivariate Gamma Family Function: Mathai and Moschopoulos (1992)*

Description

Estimate the scale parameter and shape parameters of the Mathai and Moschopoulos (1992) multivariate gamma distribution by maximum likelihood estimation.

Usage

```
gammaff.mm(lscale = "loglink", lshape = "loglink",
           iscale = NULL, ishape = NULL, imethod = 1,
           eq.shapes = FALSE, sh.byrow = TRUE, zero = "shape")
```

Arguments

`lscale, lshape` Link functions applied to the (positive) parameters b, and s_1, ..., s_Q respectively. See `Links` for more choices. In the future, `lshapes` might be used instead; of course, this link applies to all the shape parameters.

`iscale, ishape, sh.byrow` Optional initial values. The default is to compute them internally. Argument `sh.byrow` is fed into `byrow` in `matrix` and concerns the ordering of the initial shape parameters; a matrix of dimension n by Q is ultimately constructed. See also `CommonVGAMffArguments`.

`eq.shapes` Logical. Constrain the shape parameters to be equal? See also `CommonVGAMffArguments`.

`imethod, zero` See `CommonVGAMffArguments`.

Details

This distribution has the bivariate gamma distribution `bigamma.mckay` as a special case. Let $Q > 1$ be the number of columns of the response matrix `y`. Then the joint probability density function is given by

$$f(y_1,\ldots,y_Q;b,s_1,\ldots,s_Q) = y_1^{s_1}(y_2-y_1)^{s_2}\cdots(y_Q-y_{Q-1})^{s_Q}\exp(-y_Q/b)/[b^{s_Q^*}\Gamma(s_1)\cdots\Gamma(s_Q)]$$

for $b > 0$, $s_1 > 0, \ldots, s_Q > 0$ and $0 < y_1 < y_2 < \cdots < y_Q < \infty$. Also, $s_Q^* = s_1 + \cdots + s_Q$. Here, Γ is the `gamma` function, By default, the linear/additive predictors are $\eta_1 = \log(b)$, $\eta_2 = \log(s_1)$, ..., $\eta_M = \log(s_Q)$. Hence $Q = M - 1$. The marginal distributions are gamma, with shape parameters s_1 up to s_Q, but they have a common scale parameter b.

The fitted value returned is a matrix with columns equalling their respective means; for column j it is sum(shape[1:j]) * scale.

The correlations are always positive; for columns j and k with $j < k$, the correlation is sqrt(sum(shape[1:j]) /sum(shape[1:k])). Hence the variance of column j is sum(shape[1:j]) * scale^2.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm and vgam.

Note

The response must be a matrix with at least two columns. Apart from the first column, the differences between a column and its LHS adjacent column must all be positive. That is, each row must be strictly increasing.

Author(s)

T. W. Yee

References

Mathai, A. M. and Moschopoulos, P. G. (1992). A form of multivariate gamma distribution. *Ann. Inst. Statist. Math.*, **44**, 97–106.

See Also

bigamma.mckay, gammaff.

Examples

```
## Not run:
data("mbflood", package = "VGAMdata")
mbflood <- transform(mbflood, VdivD = V / D)
fit <- vglm(cbind(Q, y2 = Q + VdivD) ~ 1,
            gammaff.mm, trace = TRUE, data = mbflood)
coef(fit, matrix = TRUE)
Coef(fit)
vcov(fit)
colMeans(depvar(fit))  # Check moments
head(fitted(fit), 1)

## End(Not run)
```

`gammahyperbola` *Gamma Hyperbola Bivariate Distribution*

Description

Estimate the parameter of a gamma hyperbola bivariate distribution by maximum likelihood estimation.

Usage

```
gammahyperbola(ltheta = "loglink", itheta = NULL, expected = FALSE)
```

Arguments

`ltheta`	Link function applied to the (positive) parameter θ. See `Links` for more choices.
`itheta`	Initial value for the parameter. The default is to estimate it internally.
`expected`	Logical. `FALSE` means the Newton-Raphson (using the observed information matrix) algorithm, otherwise the expected information matrix is used (Fisher scoring algorithm).

Details

The joint probability density function is given by

$$f(y_1, y_2) = \exp(-e^{-\theta} y_1/\theta - \theta y_2)$$

for $\theta > 0$, $y_1 > 0$, $y_2 > 1$. The random variables Y_1 and Y_2 are independent. The marginal distribution of Y_1 is an exponential distribution with rate parameter $\exp(-\theta)/\theta$. The marginal distribution of Y_2 is an exponential distribution that has been shifted to the right by 1 and with rate parameter θ. The fitted values are stored in a two-column matrix with the marginal means, which are $\theta \exp(\theta)$ and $1 + 1/\theta$.

The default algorithm is Newton-Raphson because Fisher scoring tends to be much slower for this distribution.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Note

The response must be a two-column matrix.

Author(s)

T. W. Yee

References

Reid, N. (2003). Asymptotics and the theory of inference. *Annals of Statistics*, **31**, 1695–1731.

See Also

exponential.

Examples

```
gdata <- data.frame(x2 = runif(nn <- 1000))
gdata <- transform(gdata, theta = exp(-2 + x2))
gdata <- transform(gdata, y1 = rexp(nn, rate = exp(-theta)/theta),
                          y2 = rexp(nn, rate = theta) + 1)
fit <- vglm(cbind(y1, y2) ~ x2, gammahyperbola(expected = TRUE), data = gdata)
coef(fit, matrix = TRUE)
Coef(fit)
head(fitted(fit))
summary(fit)
```

---

gammaR *2-parameter Gamma Regression Family Function*

Description

Estimates the 2-parameter gamma distribution by maximum likelihood estimation.

Usage

```
gammaR(lrate = "loglink", lshape = "loglink", irate = NULL,
       ishape = NULL, lss = TRUE, zero = "shape")
```

Arguments

`lrate, lshape`	Link functions applied to the (positive) *rate* and *shape* parameters. See Links for more choices.
`irate, ishape`	Optional initial values for *rate* and *shape*. A `NULL` means a value is computed internally. If a failure to converge occurs, try using these arguments.
`zero, lss`	Details at CommonVGAMffArguments.

Details

The density function is given by

$$f(y; rate, shape) = \exp(-rate \times y) \times y^{shape-1} \times rate^{shape} / \Gamma(shape)$$

for $shape > 0$, $rate > 0$ and $y > 0$. Here, $\Gamma(shape)$ is the gamma function, as in gamma. The mean of Y is $\mu = shape/rate$ (returned as the fitted values) with variance $\sigma^2 = \mu^2/shape = shape/rate^2$. By default, the two linear/additive predictors are $\eta_1 = \log(rate)$ and $\eta_2 = \log(shape)$.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm and vgam.

Note

The parameters $rate$ and $shape$ match with the arguments rate and shape of rgamma. The order of the arguments agree too. Here, $scale = 1/rate$ is used, so one can use negloglink. Multiple responses are handled.

If $rate = 1$ use the family function gamma1 to estimate $shape$.

The reciprocal of a 2-parameter gamma random variate has an *inverse gamma* distribution. One might write a **VGAM** family function called invgammaR() to estimate this, but for now, just feed in the reciprocal of the response.

Author(s)

T. W. Yee

References

Most standard texts on statistical distributions describe the 2-parameter gamma distribution, e.g.,

Forbes, C., Evans, M., Hastings, N. and Peacock, B. (2011). *Statistical Distributions*, Hoboken, NJ, USA: John Wiley and Sons, Fourth edition.

See Also

gamma1 for the 1-parameter gamma distribution, gamma2 for another parameterization of the 2-parameter gamma distribution, bigamma.mckay for *a* bivariate gamma distribution, gammaff.mm for another, expexpff, simulate.vlm, rgamma, negloglink.

Examples

```
# Essentially a 1-parameter gamma
gdata <- data.frame(y1 = rgamma(n <- 100, shape =  exp(1)))
fit1 <- vglm(y1 ~ 1, gamma1, data = gdata, trace = TRUE)
fit2 <- vglm(y1 ~ 1, gammaR, data = gdata, trace = TRUE, crit = "coef")
coef(fit2, matrix = TRUE)
Coef(fit2)

# Essentially a 2-parameter gamma
gdata <- data.frame(y2 = rgamma(n = 500, rate = exp(1), shape = exp(2)))
fit2 <- vglm(y2 ~ 1, gammaR, data = gdata, trace = TRUE, crit = "coef")
coef(fit2, matrix = TRUE)
Coef(fit2)
summary(fit2)
```

garma *GARMA (Generalized Autoregressive Moving-Average) Models*

Description

Fits GARMA models to time series data.

Usage

```
garma(link = "identitylink", p.ar.lag = 1, q.ma.lag = 0,
      coefstart = NULL, step = 1)
```

Arguments

`link`	Link function applied to the mean response. The default is suitable for continuous responses. The link `loglink` should be chosen if the data are counts. The link `reciprocal` can be chosen if the data are counts and the variance assumed for this is μ^2. The links `logitlink`, `probitlink`, `clogloglink`, and `cauchitlink` are supported and suitable for binary responses. Note that when the log or logit link is chosen: for log and logit, zero values can be replaced by `bvalue`. See `loglink` and `logitlink` etc. for specific information about each link function.
`p.ar.lag`	A positive integer, the lag for the autoregressive component. Called p below.
`q.ma.lag`	A non-negative integer, the lag for the moving-average component. Called q below.
`coefstart`	Starting values for the coefficients. Assigning this argument is highly recommended. For technical reasons, the argument `coefstart` in `vglm` cannot be used.
`step`	Numeric. Step length, e.g., `0.5` means half-stepsizing.

Details

This function draws heavily on Benjamin *et al.* (1998). See also Benjamin *et al.* (2003). GARMA models extend the ARMA time series model to generalized responses in the exponential family, e.g., Poisson counts, binary responses. Currently, this function is rudimentary and can handle only certain continuous, count and binary responses only. The user must choose an appropriate link for the `link` argument.

The GARMA(p, q) model is defined by firstly having a response belonging to the exponential family

$$f(y_t|D_t) = \exp\left\{\frac{y_t\theta_t - b(\theta_t)}{\phi/A_t} + c(y_t, \phi/A_t)\right\}$$

where θ_t and ϕ are the canonical and scale parameters respectively, and A_t are known prior weights. The mean $\mu_t = E(Y_t|D_t) = b'(\theta_t)$ is related to the linear predictor η_t by the link function g.

Here, $D_t = \{x_t, \ldots, x_1, y_{t-1}, \ldots, y_1, \mu_{t-1}, \ldots, \mu_1\}$ is the previous information set. Secondly, the GARMA(p, q) model is defined by

$$g(\mu_t) = \eta_t = x_t^T \beta + \sum_{k=1}^{p} \phi_k (g(y_{t-k}) - x_{t-k}^T \beta) + \sum_{k=1}^{q} \theta_k (g(y_{t-k}) - \eta_{t-k}).$$

Parameter vectors β, ϕ and θ are estimated by maximum likelihood.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm.

Warning

This **VGAM** family function is 'non-standard' in that the model does need some coercing to get it into the VGLM framework. Special code is required to get it running. A consequence is that some methods functions may give wrong results when applied to the fitted object.

Note

This function is unpolished and is requires *lots* of improvements. In particular, initialization is *very poor*. Results appear *very* sensitive to quality of initial values. A limited amount of experience has shown that half-stepsizing is often needed for convergence, therefore choosing `crit = "coef"` is not recommended.

Overdispersion is not handled. For binomial responses it is currently best to input a vector of 1s and 0s rather than the `cbind(successes, failures)` because the initialize slot is rudimentary.

Author(s)

T. W. Yee

References

Benjamin, M. A., Rigby, R. A. and Stasinopoulos, M. D. (1998). Fitting Non-Gaussian Time Series Models. Pages 191–196 in: *Proceedings in Computational Statistics COMPSTAT 1998* by Payne, R. and P. J. Green. Physica-Verlag.

Benjamin, M. A., Rigby, R. A. and Stasinopoulos, M. D. (2003). Generalized Autoregressive Moving Average Models. *Journal of the American Statistical Association*, **98**: 214–223.

Zeger, S. L. and Qaqish, B. (1988). Markov regression models for time series: a quasi-likelihood approach. *Biometrics*, **44**: 1019–1031.

Examples

```
gdata <- data.frame(interspike = c(68, 41, 82, 66, 101, 66, 57,  41,  27, 78,
59, 73,  6, 44,  72, 66, 59,  60,  39, 52,
50, 29, 30, 56,  76, 55, 73, 104, 104, 52,
25, 33, 20, 60,  47,  6, 47,  22,  35, 30,
29, 58, 24, 34,  36, 34,  6,  19,  28, 16,
```

```
36, 33, 12, 26,  36, 39, 24,  14,  28, 13,
 2, 30, 18, 17,  28,  9, 28,  20,  17, 12,
19, 18, 14, 23,  18, 22, 18,  19,  26, 27,
23, 24, 35, 22,  29, 28, 17,  30,  34, 17,
20, 49, 29, 35,  49, 25, 55,  42,  29, 16))  # See Zeger and Qaqish (1988)
gdata <- transform(gdata, spikenum = seq(interspike))
bvalue <- 0.1  # .Machine$double.xmin # Boundary value
fit <- vglm(interspike ~ 1, trace = TRUE, data = gdata,
            garma(loglink(bvalue = bvalue),
                  p = 2, coefstart = c(4, 0.3, 0.4)))
summary(fit)
coef(fit, matrix = TRUE)
Coef(fit)  # A bug here
## Not run:  with(gdata, plot(interspike, ylim = c(0, 120), las = 1,
     xlab = "Spike Number", ylab = "Inter-Spike Time (ms)", col = "blue"))
with(gdata, lines(spikenum[-(1:fit@misc$plag)], fitted(fit), col = "orange"))
abline(h = mean(with(gdata, interspike)), lty = "dashed", col = "gray")
## End(Not run)
```

GenbetaII *The Generalized Beta II Distribution*

Description

Density for the generalized beta II distribution with shape parameters a and p and q, and scale parameter scale.

Usage

```
dgenbetaII(x, scale = 1, shape1.a, shape2.p, shape3.q, log = FALSE)
```

Arguments

x	vector of quantiles.
shape1.a, shape2.p, shape3.q	positive shape parameters.
scale	positive scale parameter.
log	Logical. If log = TRUE then the logarithm of the density is returned.

Details

See genbetaII, which is the **VGAM** family function for estimating the parameters by maximum likelihood estimation. Several distributions, such as the Singh-Maddala, are special case of this flexible 4-parameter distribution. The product of shape1.a and shape2.p determines the behaviour of the density at the origin.

Value

dgenbetaII gives the density.

Author(s)

T. W. Yee

References

Kleiber, C. and Kotz, S. (2003). *Statistical Size Distributions in Economics and Actuarial Sciences*, Hoboken, NJ, USA: Wiley-Interscience.

See Also

genbetaII.

Examples

```
dgenbetaII(0, shape1.a = 1/4, shape2.p = 4, shape3.q = 3)
dgenbetaII(0, shape1.a = 1/4, shape2.p = 2, shape3.q = 3)
dgenbetaII(0, shape1.a = 1/4, shape2.p = 8, shape3.q = 3)
```

---

genbetaII *Generalized Beta Distribution of the Second Kind*

Description

Maximum likelihood estimation of the 4-parameter generalized beta II distribution.

Usage

```
genbetaII(lscale = "loglink", lshape1.a = "loglink",
     lshape2.p = "loglink", lshape3.q = "loglink",
     iscale = NULL, ishape1.a = NULL,
     ishape2.p = NULL, ishape3.q = NULL, lss = TRUE,
     gscale = exp(-5:5), gshape1.a = exp(-5:5),
     gshape2.p = exp(-5:5), gshape3.q = exp(-5:5), zero = "shape")
```

Arguments

`lss` See CommonVGAMffArguments for important information.

`lshape1.a, lscale, lshape2.p, lshape3.q`
Parameter link functions applied to the shape parameter `a`, scale parameter `scale`, shape parameter `p`, and shape parameter `q`. All four parameters are positive. See Links for more choices.

`iscale, ishape1.a, ishape2.p, ishape3.q`
Optional initial values for the parameters. A `NULL` means a value is computed internally using the arguments `gscale`, `gshape1.a`, etc.

`gscale, gshape1.a, gshape2.p, gshape3.q`
See CommonVGAMffArguments for information. Replaced by `iscale`, `ishape1.a` etc. if given.

`zero` The default is to set all the shape parameters to be intercept-only. See CommonVGAMffArguments for information.

Details

This distribution is most useful for unifying a substantial number of size distributions. For example, the Singh-Maddala, Dagum, Fisk (log-logistic), Lomax (Pareto type II), inverse Lomax, beta distribution of the second kind distributions are all special cases. Full details can be found in Kleiber and Kotz (2003), and Brazauskas (2002). The argument names given here are used by other families that are special cases of this family. Fisher scoring is used here and for the special cases too.

The 4-parameter generalized beta II distribution has density

$$f(y) = ay^{ap-1}/[b^{ap}B(p,q)\{1+(y/b)^a\}^{p+q}]$$

for $a > 0$, $b > 0$, $p > 0$, $q > 0$, $y \geq 0$. Here B is the beta function, and b is the scale parameter `scale`, while the others are shape parameters. The mean is

$$E(Y) = b\,\Gamma(p+1/a)\,\Gamma(q-1/a)/(\Gamma(p)\,\Gamma(q))$$

provided $-ap < 1 < aq$; these are returned as the fitted values.

This family function handles multiple responses.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Warning

This distribution is very flexible and it is not generally recommended to use this family function when the sample size is small—numerical problems easily occur with small samples. Probably several hundred observations at least are needed in order to estimate the parameters with any level of confidence. Neither is the inclusion of covariates recommended at all—not unless there are several thousand observations. The mean is finite only when $-ap < 1 < aq$, and this can be easily violated by the parameter estimates for small sample sizes. Try fitting some of the special cases of this distribution (e.g., `sinmad`, `fisk`, etc.) first, and then possibly use those models for initial values for this distribution.

Note

The default is to use a grid search with respect to all four parameters; this is quite costly and is time consuming. If the self-starting initial values fail, try experimenting with the initial value arguments. Also, the constraint $-ap < 1 < aq$ may be violated as the iterations progress so it pays to monitor convergence, e.g., set `trace = TRUE`. Successful convergence depends on having very good initial values. This is rather difficult for this distribution so that a grid search is conducted by default. One suggestion for increasing the estimation reliability is to set `stepsize = 0.5` and `maxit = 100`; see `vglm.control`.

Author(s)

T. W. Yee, with help from Victor Miranda.

References

Kleiber, C. and Kotz, S. (2003). *Statistical Size Distributions in Economics and Actuarial Sciences*, Hoboken, NJ, USA: Wiley-Interscience.

Brazauskas, V. (2002). Fisher information matrix for the Feller-Pareto distribution. *Statistics & Probability Letters*, **59**, 159–167.

See Also

`dgenbetaII`, `betaff`, `betaII`, `dagum`, `sinmad`, `fisk`, `lomax`, `inv.lomax`, `paralogistic`, `inv.paralogistic`, `lino`, `CommonVGAMffArguments`, `vglm.control`.

Examples

```
## Not run:
gdata <- data.frame(y = rsinmad(3000, shape1 = exp(1), scale = exp(2),
                                shape3 = exp(1)))  # A special case!
fit <- vglm(y ~ 1, genbetaII(lss = FALSE), data = gdata, trace = TRUE)
fit <- vglm(y ~ 1, data = gdata, trace = TRUE,
            genbetaII(ishape1.a = 3, iscale = 7, ishape3.q = 2.3))
coef(fit, matrix = TRUE)
Coef(fit)
summary(fit)

## End(Not run)
```

`gengamma.stacy` *Generalized Gamma distribution family function*

Description

Estimation of the 3-parameter generalized gamma distribution proposed by Stacy (1962).

Usage

```
gengamma.stacy(lscale = "loglink", ld = "loglink", lk = "loglink",
    iscale = NULL, id = NULL, ik = NULL, imethod = 1,
    gscale.mux = exp((-4:4)/2), gshape1.d = exp((-5:5)/2),
    gshape2.k = exp((-5:5)/2), probs.y = 0.3, zero = c("d", "k"))
```

Arguments

`lscale, ld, lk` Parameter link function applied to each of the positive parameters b, d and k, respectively. See `Links` for more choices.

`iscale, id, ik` Initial value for b, d and k, respectively. The defaults mean an initial value is determined internally for each.

`gscale.mux, gshape1.d, gshape2.k`
: See `CommonVGAMffArguments` for information. Replaced by `iscale`, `id` etc. if given.

`imethod, probs.y, zero`
: See CommonVGAMffArguments for information.

Details

The probability density function can be written

$$f(y; b, d, k) = db^{-dk} y^{dk-1} \exp[-(y/b)^d]/\Gamma(k)$$

for scale parameter $b > 0$, and Weibull-type shape parameter $d > 0$, gamma-type shape parameter $k > 0$, and $y > 0$. The mean of Y is $b \times \Gamma(k + 1/d)/\Gamma(k)$ (returned as the fitted values), which equals bk if $d = 1$.

There are many special cases, as given in Table 1 of Stacey and Mihram (1965). In the following, the parameters are in the order b, d, k. The special cases are: Exponential $f(y; b, 1, 1)$, Gamma $f(y; b, 1, k)$, Weibull $f(y; b, d, 1)$, Chi Squared $f(y; 2, 1, a/2)$ with a degrees of freedom, Chi $f(y; \sqrt{2}, 2, a/2)$ with a degrees of freedom, Half-normal $f(y; \sqrt{2}, 2, 1/2)$, Circular normal $f(y; \sqrt{2}, 2, 1)$, Spherical normal $f(y; \sqrt{2}, 2, 3/2)$, Rayleigh $f(y; c\sqrt{2}, 2, 1)$ where $c > 0$. Also the log-normal distribution corresponds to when `k = Inf`.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Warning

Several authors have considered maximum likelihood estimation for the generalized gamma distribution and have found that the Newton-Raphson algorithm does not work very well and that the existence of solutions to the log-likelihood equations is sometimes in doubt. Although Fisher scoring is used here, it is likely that the same problems will be encountered. It appears that large samples are required, for example, the estimator of k became asymptotically normal only with 400 or more observations. It is not uncommon for maximum likelihood estimates to fail to converge even with two or three hundred observations. With covariates, even more observations are needed to increase the chances of convergence. Using covariates is not advised unless the sample size is at least a few thousand, and even if so, modelling 1 or 2 parameters as intercept-only is a very good idea (e.g., `zero = 2:3`). Monitoring convergence is also a very good idea (e.g., set `trace = TRUE`). Half-stepping is not uncommon, and if this occurs, then the results should be viewed with more suspicion.

Note

The notation used here differs from Stacy (1962) and Prentice (1974). Poor initial values may result in failure to converge so if there are covariates and there are convergence problems, try using or checking the `zero` argument (e.g., `zero = 2:3`) or the `ik` argument or the `imethod` argument, etc.

Author(s)

T. W. Yee

References

Stacy, E. W. (1962). A generalization of the gamma distribution. *Annals of Mathematical Statistics*, **33**(3), 1187–1192.

Stacy, E. W. and Mihram, G. A. (1965). Parameter estimation for a generalized gamma distribution. *Technometrics*, **7**, 349–358.

Prentice, R. L. (1974). A log gamma model and its maximum likelihood estimation. *Biometrika*, **61**, 539–544.

See Also

`rgengamma.stacy`, `gamma1`, `gamma2`, `prentice74`, `simulate.vlm`, `chisq`, `lognormal`, `rayleigh`, `weibullR`.

Examples

```
k <- exp(-1); Scale <- exp(1); dd <- exp(0.5); set.seed(1)
gdata <- data.frame(y = rgamma(2000, shape = k, scale = Scale))
gfit <- vglm(y ~ 1, gengamma.stacy, data = gdata, trace = TRUE)
coef(gfit, matrix = TRUE)
```

`gengammaUC` *Generalized Gamma Distribution*

Description

Density, distribution function, quantile function and random generation for the generalized gamma distribution with scale parameter `scale`, and parameters `d` and `k`.

Usage

```
dgengamma.stacy(x, scale = 1, d, k, log = FALSE)
pgengamma.stacy(q, scale = 1, d, k,
                lower.tail = TRUE, log.p = FALSE)
qgengamma.stacy(p, scale = 1, d, k,
                lower.tail = TRUE, log.p = FALSE)
rgengamma.stacy(n, scale = 1, d, k)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. Same as in `runif`.
`scale`	the (positive) scale parameter b.
`d, k`	the (positive) parameters d and k. Both can be thought of as shape parameters, where d is of the Weibull-type and k is of the gamma-type.

log Logical. If log = TRUE then the logarithm of the density is returned.

lower.tail, log.p
Same meaning as in pnorm or qnorm.

Details

See gengamma.stacy, the **VGAM** family function for estimating the generalized gamma distribution by maximum likelihood estimation, for formulae and other details. Apart from n, all the above arguments may be vectors and are recyled to the appropriate length if necessary.

Value

dgengamma.stacy gives the density, pgengamma.stacy gives the distribution function, qgengamma.stacy gives the quantile function, and rgengamma.stacy generates random deviates.

Author(s)

T. W. Yee and Kai Huang

References

Stacy, E. W. and Mihram, G. A. (1965). Parameter estimation for a generalized gamma distribution. *Technometrics*, **7**, 349–358.

See Also

gengamma.stacy.

Examples

```
## Not run:  x <- seq(0, 14, by = 0.01); d <- 1.5; Scale <- 2; k <- 6
plot(x, dgengamma.stacy(x, Scale, d = d, k = k), type = "l",
     col = "blue", ylim = 0:1,
     main = "Blue is density, orange is the CDF",
     sub = "Purple are 5,10,...,95 percentiles", las = 1, ylab = "")
abline(h = 0, col = "blue", lty = 2)
lines(qgengamma.stacy(seq(0.05, 0.95, by = 0.05), Scale, d = d, k = k),
      dgengamma.stacy(qgengamma.stacy(seq(0.05, 0.95, by = 0.05),
                                      Scale, d = d, k = k),
            Scale, d = d, k = k), col = "purple", lty = 3, type = "h")
lines(x, pgengamma.stacy(x, Scale, d = d, k = k), col = "orange")
abline(h = 0, lty = 2)
## End(Not run)
```

Genpois0 *Generalized Poisson Distribution (Original Parameterization)*

Description

Density, distribution function, quantile function and random generation for the original parameterization of the generalized Poisson distribution.

Usage

```
dgenpois0(x, theta, lambda = 0, log = FALSE)
pgenpois0(q, theta, lambda = 0, lower.tail = TRUE)
qgenpois0(p, theta, lambda = 0)
rgenpois0(n, theta, lambda = 0, algorithm = c("qgenpois0",
          "inv", "bup","chdn", "napp", "bran"))
```

Arguments

x, q	Vector of quantiles.
p	Vector of probabilities.
n	Similar to runif.
theta, lambda	See genpoisson0. The default value of lambda corresponds to an ordinary Poisson distribution. *Nonnegative* values of lambda are currently required.
lower.tail, log	Similar to Poisson.
algorithm	Character. Six choices are available, standing for the *qgenpois0*, *inversion*, *build-up*, *chop-down*, *normal approximation* and *branching* methods. The first one is the default and calls qgenpois0 with runif as its first argument. The value inputted may be abbreviated, e.g., alg = "n". The last 5 algorithms are a direct implementation of Demirtas (2017) and the relative performance of the algorithms are described there—however, the vectorization here may render the comments on relative speed as no longer holding.

Details

Most of the background to these functions are given in genpoisson0. Some warnings relevant to this distribution are given there. The complicated range of the parameter lambda when negative is no longer supported because the distribution is not normalized. For other GPD variants see Genpois1.

Value

dgenpois0 gives the density, pgenpois0 gives the distribution function, qgenpois0 gives the quantile function, and rgenpois generates random deviates. For some of these functions such as dgenpois0 and pgenpois0 the value NaN is returned for elements not satisfying the parameter restrictions, e.g., if $\lambda > 1$. For some of these functions such as rgenpois0 the input must not contain NAs or NaNs, etc. since the implemented algorithms are fragile.

Warning

These have not been tested thoroughly.

For pgentpois0() mapply is called with 0:q as input, hence will be very slow and memory-hungry for large values of q. Likewise qgentpois0() and rgentpois0() may suffer from the same limitations.

Note

For rgentpois0(): (1). "inv", "bup" and "chdn" appear similar and seem to work okay. (2). "napp" works only when theta is large, away from 0. It suffers from 0-inflation. (3). "bran" has a relatively heavy RHS tail and requires positive lambda. More details can be found in Famoye (1997) and Demirtas (2017).

The function dgenpois0 uses lfactorial, which equals Inf when x is approximately 1e306 on many machines. So the density is returned as 0 in very extreme cases; see .Machine.

Author(s)

T. W. Yee. For rgenpois0() the last 5 algorithms are based on code written in H. Demirtas (2017) and vectorized by T. W. Yee; but the "bran" algorithm was rewritten from Famoye (1997).

References

Demirtas, H. (2017). On accurate and precise generation of generalized Poisson variates. *Communications in Statistics—Simulation and Computation*, **46**, 489–499.

Famoye, F. (1997). Generalized Poisson random variate generation. *Amer. J. Mathematical and Management Sciences*, **17**, 219–237.

See Also

genpoisson0, Genpois1, dpois.

Examples

```
sum(dgenpois0(0:1000, theta = 2, lambda = 0.5))
## Not run: theta <- 2; lambda <- 0.2; y <- 0:10
proby <- dgenpois0(y, theta = theta, lambda = lambda, log = FALSE)
plot(y, proby, type = "h", col = "blue", lwd = 2, ylab = "Pr(Y=y)",
     main = paste0("Y ~ GP-0(theta=", theta, ", lambda=",
                   lambda, ")"), las = 1, ylim = c(0, 0.3),
     sub = "Orange is the Poisson probability function")
lines(y + 0.1, dpois(y, theta), type = "h", lwd = 2, col = "orange")
## End(Not run)
```

Genpois1 *Generalized Poisson Distribution (GP-1 and GP-2 Parameterizations of the Mean)*

Description

Density, distribution function, quantile function and random generation for two parameterizations (GP-1 and GP-2) of the generalized Poisson distribution of the mean.

Usage

```
dgenpois1(x, meanpar, dispind = 1, log = FALSE)
pgenpois1(q, meanpar, dispind = 1, lower.tail = TRUE)
qgenpois1(p, meanpar, dispind = 1)
rgenpois1(n, meanpar, dispind = 1)
dgenpois2(x, meanpar, disppar = 0, log = FALSE)
pgenpois2(q, meanpar, disppar = 0, lower.tail = TRUE)
qgenpois2(p, meanpar, disppar = 0)
rgenpois2(n, meanpar, disppar = 0)
```

Arguments

`x, q` Vector of quantiles.

`p` Vector of probabilities.

`n` Similar to `runif`.

`meanpar, dispind` The mean and dispersion index (index of dispersion), which are the two parameters for the GP-1. The mean is positive while the `dispind` is ≥ 1. The default value of `dispind` corresponds to an ordinary Poisson distribution.

`disppar` The dispersion parameter for the GP-2: `disppar` ≥ 0. The default value of `disppar` corresponds to an ordinary Poisson distribution.

`lower.tail, log` See `Genpois0`.

Details

These are wrapper functions for those in `Genpois0`. The first parameter is the mean, therefore both the GP-1 and GP-2 are recommended for regression and can be compared somewhat to `poissonff` and `negbinomial`. The variance of a GP-1 is $\mu\varphi$ where $\varphi = 1/(1-\lambda)^2$ is `dispind`.

The variance of a GP-2 is $\mu(1+\alpha\mu)^2$ where $\theta = \mu/(1+\alpha\mu)$, $\lambda = \alpha\mu/(1+\alpha\mu)$, and is α is the dispersion parameter `disppar`. Thus the variance is linear with respect to the mean for GP-1 while the variance is cubic with respect to the mean for GP-2.

Recall that the *index of dispersion* (also known as the *dispersion index*) is the ratio of the variance and the mean. Also, $\mu = \theta/(1-\lambda)$ in the original formulation with variance $\theta/(1-\lambda)^3$. The GP-1 is due to Consul and Famoye (1992). The GP-2 is due to Wang and Famoye (1997).

Value

dgenpois1 and dgenpois2 give the density, pgenpois1 and dgenpois2 give the distribution function, qgenpois1 and dgenpois2 give the quantile function, and rgenpois1 and dgenpois2 generate random deviates. See Genpois0 for more information.

Warning

Genpois0 has warnings that should be heeded.

Author(s)

T. W. Yee.

References

Consul, P. C. and Famoye, F. (1992). Generalized Poisson regression model. *Comm. Statist.—Theory and Meth.*, **2**, 89–109.

Wang, W. and Famoye, F. (1997). Modeling household fertility decisions with generalized Poisson regression. *J. Population Econom.*, **10**, 273–283.

See Also

Genpois0.

Examples

```
sum(dgenpois1(0:1000, meanpar = 5, dispind = 2))
## Not run: dispind <- 5; meanpar <- 5; y <- 0:15
proby <- dgenpois1(y, meanpar = meanpar, dispind)
plot(y, proby, type = "h", col = "blue", lwd = 2, ylab = "P[Y=y]",
     main = paste0("Y ~ GP-1(meanpar=", meanpar, ", dispind=",
                   dispind, ")"), las = 1, ylim = c(0, 0.3),
     sub = "Orange is the Poisson probability function")
lines(y + 0.1, dpois(y, meanpar), type = "h", lwd = 2, col = "orange")
## End(Not run)
```

---

genpoisson0 *Generalized Poisson Regression (Original Parameterization)*

Description

Estimation of the two-parameter generalized Poisson distribution (original parameterization).

Usage

```
genpoisson0(ltheta = "loglink", llambda = "logitlink",
            itheta = NULL, ilambda = NULL, imethod = c(1, 1),
            ishrinkage = 0.95, glambda = ppoints(5),
            parallel = FALSE, zero = "lambda")
```

Arguments

`ltheta, llambda`
: Parameter link functions for θ and λ. See `Links` for more choices. In theory the λ parameter is allowed to be negative to handle underdispersion, however this is no longer supported, hence $0 < \lambda < 1$. The θ parameter is positive, therefore the default is the log link.

`itheta, ilambda`
: Optional initial values for λ and θ. The default is to choose values internally.

`imethod`
: See `CommonVGAMffArguments` for information. Each value is an integer `1` or `2` or `3` which specifies the initialization method for each of the parameters. If failure to converge occurs try another value and/or else specify a value for `ilambda` and/or `itheta`. The argument is recycled to length 2, and the first value corresponds to `theta`, etc.

`ishrinkage, zero`
: See `CommonVGAMffArguments` for information.

`glambda, parallel`
: See `CommonVGAMffArguments` for information. Argument `glambda` is similar to `gsigma` there and is currently used only if `imethod[2] = 1`.

Details

The generalized Poisson distribution (GPD) was proposed by Consul and Jain (1973), and it has PMF

$$f(y) = \theta(\theta + \lambda y)^{y-1} \exp(-\theta - \lambda y)/y!$$

for $0 < \theta$ and $y = 0, 1, 2, \ldots$. Theoretically, $\max(-1, -\theta/m) \leq \lambda \leq 1$ where m (≥ 4) is the greatest positive integer satisfying $\theta + m\lambda > 0$ when $\lambda < 0$ [and then $Pr(Y = y) = 0$ for $y > m$]. However, there are problems with a negative λ such as it not being normalized, so this family function restricts λ to $(0, 1)$.

This original parameterization is called the GP-0 by **VGAM**, partly because there are two other common parameterizations called the GP-1 and GP-2 (see Yang et al. (2009)), `genpoisson1` and `genpoisson2`) that are more suitable for regression. However, `genpoisson()` has been simplified to `genpoisson0` by only handling positive parameters, hence only overdispersion relative to the Poisson is accommodated. Some of the reasons for this are described in Scollnik (1998), e.g., the probabilities do not sum to unity when `lambda` is negative. To simply things, **VGAM** 1.1-4 and later will only handle positive `lambda`.

An ordinary Poisson distribution corresponds to $\lambda = 0$. The mean (returned as the fitted values) is $E(Y) = \theta/(1 - \lambda)$ and the variance is $\theta/(1 - \lambda)^3$ so that the variance is proportional to the mean, just like the NB-1 and quasi-Poisson.

For more information see Consul and Famoye (2006) for a summary and Consul (1989) for more details.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Warning

Although this family function is far less fragile compared to what used to be called genpoisson() it is still a good idea to monitor convergence because equidispersion may result in numerical problems; try poissonff instead. And underdispersed data will definitely result in numerical problems and warnings; try quasipoisson instead.

Note

This family function replaces genpoisson(), and some of the major changes are: (i) the swapping of the linear predictors; (ii) the change from rhobitlink to logitlink in llambda to reflect the no longer handling of underdispersion; (iii) proper Fisher scoring is implemented to give improved convergence.

Notationally, and in the literature too, don't get confused because theta (and not lambda) here really matches more closely with lambda of dpois.

This family function handles multiple responses. This distribution is potentially useful for dispersion modelling. Convergence and numerical problems may occur when lambda becomes very close to 0 or 1.

Author(s)

T. W. Yee. Easton Huch derived the EIM and it has been implemented in the weights slot.

References

Consul, P. C. and Jain, G. C. (1973). A generalization of the Poisson distribution. *Technometrics*, **15**, 791–799.

Consul, P. C. and Famoye, F. (2006). *Lagrangian Probability Distributions*, Boston, USA: Birkhauser.

Jorgensen, B. (1997). *The Theory of Dispersion Models*. London: Chapman & Hall.

Consul, P. C. (1989). *Generalized Poisson Distributions: Properties and Applications*. New York, USA: Marcel Dekker.

Yang, Z., Hardin, J. W., Addy, C. L. (2009). A score test for overdispersion in Poisson regression based on the generalized Poisson-2 model. *J. Statist. Plann. Infer.*, **139**, 1514–1521.

Yee, T. W. (2020). On generalized Poisson regression. *In preparation*.

See Also

Genpois0, genpoisson1, genpoisson2, poissonff, negbinomial, Poisson, quasipoisson.

Examples

```
gdata <- data.frame(x2 = runif(nn <- 500))
gdata <- transform(gdata, y1 = rgenpois0(nn, theta = exp(2 + x2),
                                         logitlink(1, inverse = TRUE)))
gfit0 <- vglm(y1 ~ x2, genpoisson0, data = gdata, trace = TRUE)
coef(gfit0, matrix = TRUE)
summary(gfit0)
```

genpoisson1 *Generalized Poisson Regression (GP-1 Parameterization)*

Description

Estimation of the two-parameter generalized Poisson distribution (GP-1 parameterization) which has the variance as a linear function of the mean.

Usage

```
genpoisson1(lmeanpar = "loglink", ldispind = "logloglink",
     parallel = FALSE, zero = "dispind",
     vfl = FALSE, Form2 = NULL,
     imeanpar = NULL, idispind = NULL, imethod = c(1, 1),
     ishrinkage = 0.95, gdispind = exp(1:5))
```

Arguments

`lmeanpar, ldispind`
: Parameter link functions for μ and φ. They are called the *mean par*ameter and *disp*ersion *ind*ex respectively. See `Links` for more choices. In theory the φ parameter might be allowed to be less than unity to handle underdispersion but this is not supported. The mean is positive so its default is the log link. The dispersion index is > 1 so its default is the log-log link.

`vfl, Form2`
: If `vfl = TRUE` then `Form2` should be assigned a formula having terms comprising $\eta_2 = \log\log\varphi$. This is similar to `uninormal`. See `CommonVGAMffArguments` for information.

`imeanpar, idispind`
: Optional initial values for μ and φ. The default is to choose values internally.

`imethod`
: See `CommonVGAMffArguments` for information. The argument is recycled to length 2, and the first value corresponds to μ, etc.

`ishrinkage, zero`
: See `CommonVGAMffArguments` for information.

`gdispind, parallel`
: See `CommonVGAMffArguments` for information. Argument `gdispind` is similar to `gsigma` there and is currently used only if `imethod[2] = 2`.

Details

This is a variant of the generalized Poisson distribution (GPD) and is similar to the GP-1 referred to by some writers such as Yang, et al. (2009). Compared to the original GP-0 (see `genpoisson0`) the GP-1 has $\theta = \mu/\sqrt{\varphi}$ and $\lambda = 1 - 1/\sqrt{\varphi}$ so that the variance is $\mu\varphi$. The first linear predictor by default is $\eta_1 = \log\mu$ so that the GP-1 is more suitable for regression than the GP-1.

This family function can handle only overdispersion relative to the Poisson. An ordinary Poisson distribution corresponds to $\varphi = 1$. The mean (returned as the fitted values) is $E(Y) = \mu$. For overdispersed data, this GP parameterization is a direct competitor of the NB-1 and quasi-Poisson.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Warning

See genpoisson0 for warnings relevant here, e.g., it is a good idea to monitor convergence because of equidispersion and underdispersion.

Author(s)

T. W. Yee.

See Also

Genpois1, genpoisson0, genpoisson2, poissonff, negbinomial, Poisson, quasipoisson.

Examples

```
gdata <- data.frame(x2 = runif(nn <- 500))
gdata <- transform(gdata, y1 = rgenpois1(nn, exp(2 + x2),
                              logloglink(-1, inverse = TRUE)))
gfit1 <- vglm(y1 ~ x2, genpoisson1, gdata, trace = TRUE)
coef(gfit1, matrix = TRUE)
summary(gfit1)
```

---

genpoisson2 *Generalized Poisson Regression (GP-2 Parameterization)*

Description

Estimation of the two-parameter generalized Poisson distribution (GP-2 parameterization) which has the variance as a cubic function of the mean.

Usage

```
genpoisson2(lmeanpar = "loglink", ldisppar = "loglink",
    parallel = FALSE, zero = "disppar",
    vfl = FALSE, oparallel = FALSE,
    imeanpar = NULL, idisppar = NULL, imethod = c(1, 1),
    ishrinkage = 0.95, gdisppar = exp(1:5))
```

Arguments

`lmeanpar, ldisppar`
: Parameter link functions for μ and α. They are called the *mean* and *disp*ersion *par*ameters respectively. See `Links` for more choices. In theory the α parameter might be allowed to be negative to handle underdispersion but this is not supported. All parameters are positive, therefore the defaults are the log link.

`imeanpar, idisppar`
: Optional initial values for μ and α. The default is to choose values internally.

`vfl, oparallel`
: Argument `oparallel` is similar to `parallel` but uses `rbind(1, -1)` instead. If `vfl = TRUE` then `oparallel` should be assigned a formula having terms comprising $\eta_1 = \log\mu$, and then the other terms in the main formula are for $\eta_2 = \log\alpha$. See `CommonVGAMffArguments` for information.

`imethod`
: See `CommonVGAMffArguments` for information. The argument is recycled to length 2, and the first value corresponds to μ, etc.

`ishrinkage, zero`
: See `CommonVGAMffArguments` for information.

`gdisppar, parallel`
: See `CommonVGAMffArguments` for information. Argument `gdisppar` is similar to `gsigma` there and is currently used only if `imethod[2] = 2`.

Details

This is a variant of the generalized Poisson distribution (GPD) and called GP-2 by some writers such as Yang, et al. (2009). Compared to the original GP-0 (see `genpoisson0`) the GP-2 has $\theta = \mu/(1+\alpha\mu)$ and $\lambda = \alpha\mu/(1+\alpha\mu)$ so that the variance is $\mu(1+\alpha\mu)^2$. The first linear predictor by default is $\eta_1 = \log\mu$ so that the GP-2 is more suitable for regression than the GP-0.

This family function can handle only overdispersion relative to the Poisson. An ordinary Poisson distribution corresponds to $\alpha = 0$. The mean (returned as the fitted values) is $E(Y) = \mu$.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Warning

See `genpoisson0` for warnings relevant here, e.g., it is a good idea to monitor convergence because of equidispersion and underdispersion.

Author(s)

T. W. Yee.

References

Letac, G. and Mora, M. (1990). Natural real exponential familes with cubic variance functions. *Annals of Statistics* **18**, 1–37.

See Also

Genpois2, genpoisson0, genpoisson1, poissonff, negbinomial, Poisson, quasipoisson.

Examples

```
gdata <- data.frame(x2 = runif(nn <- 500))
gdata <- transform(gdata, y1 = rgenpois2(nn, exp(2 + x2),
                               loglink(-1, inverse = TRUE)))
gfit2 <- vglm(y1 ~ x2, genpoisson2, gdata, trace = TRUE)
coef(gfit2, matrix = TRUE)
summary(gfit2)
```

---

genray *The Generalized Rayleigh Distribution*

Description

Density, distribution function, quantile function and random generation for the generalized Rayleigh distribution.

Usage

```
dgenray(x, scale = 1, shape, log = FALSE)
pgenray(q, scale = 1, shape, lower.tail = TRUE, log.p = FALSE)
qgenray(p, scale = 1, shape, lower.tail = TRUE, log.p = FALSE)
rgenray(n, scale = 1, shape)
```

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
n	number of observations. If length(n) > 1 then the length is taken to be the number required.
scale, shape	positive scale and shape parameters.
log	Logical. If log = TRUE then the logarithm of the density is returned.
lower.tail, log.p	Same meaning as in pnorm or qnorm.

Details

See genrayleigh, the **VGAM** family function for estimating the parameters, for the formula of the probability density function and other details.

Value

dgenray gives the density, pgenray gives the distribution function, qgenray gives the quantile function, and rgenray generates random deviates.

Note

We define `scale` as the reciprocal of the scale parameter used by Kundu and Raqab (2005).

Author(s)

Kai Huang and J. G. Lauder and T. W. Yee

See Also

`genrayleigh`, `rayleigh`.

Examples

```
## Not run:
shape <- 0.5; Scale <- 1; nn <- 501
x <- seq(-0.10, 3.0, len = nn)
plot(x, dgenray(x, shape, scale = Scale), type = "l", las = 1, ylim = c(0, 1.2),
     ylab = paste("[dp]genray(shape = ", shape, ", scale = ", Scale, ")"),
     col = "blue", cex.main = 0.8,
     main = "Blue is density, orange is cumulative distribution function",
     sub = "Purple lines are the 10,20,...,90 percentiles")
lines(x, pgenray(x, shape, scale = Scale), col = "orange")
probs <- seq(0.1, 0.9, by = 0.1)
Q <- qgenray(probs, shape, scale = Scale)
lines(Q, dgenray(Q, shape, scale = Scale), col = "purple", lty = 3, type = "h")
lines(Q, pgenray(Q, shape, scale = Scale), col = "purple", lty = 3, type = "h")
abline(h = probs, col = "purple", lty = 3)
max(abs(pgenray(Q, shape, scale = Scale) - probs))  # Should be 0

## End(Not run)
```

`genrayleigh` *Generalized Rayleigh Distribution Family Function*

Description

Estimates the two parameters of the generalized Rayleigh distribution by maximum likelihood estimation.

Usage

```
genrayleigh(lscale = "loglink", lshape = "loglink",
            iscale = NULL,   ishape = NULL,
            tol12 = 1e-05, nsimEIM = 300, zero = 2)
```

Arguments

`lscale, lshape`	Link function for the two positive parameters, scale and shape. See `Links` for more choices.
`iscale, ishape`	Numeric. Optional initial values for the scale and shape parameters.
`nsimEIM, zero`	See `CommonVGAMffArguments`.
`tol12`	Numeric and positive. Tolerance for testing whether the second shape parameter is either 1 or 2. If so then the working weights need to handle these singularities.

Details

The generalized Rayleigh distribution has density function

$$f(y; b = scale, s = shape) = (2sy/b^2)e^{-(y/b)^2}(1 - e^{-(y/b)^2})^{s-1}$$

where $y > 0$ and the two parameters, b and s, are positive. The mean cannot be expressed nicely so the median is returned as the fitted values. Applications of the generalized Rayleigh distribution include modeling strength data and general lifetime data. Simulated Fisher scoring is implemented.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Note

We define `scale` as the reciprocal of the scale parameter used by Kundu and Raqab (2005).

Author(s)

J. G. Lauder and T. W. Yee

References

Kundu, D., Raqab, M. C. (2005). Generalized Rayleigh distribution: different methods of estimations. *Computational Statistics and Data Analysis*, **49**, 187–200.

See Also

`dgenray`, `rayleigh`.

Examples

```
Scale <- exp(1); shape <- exp(1)
rdata <- data.frame(y = rgenray(n = 1000, scale = Scale, shape = shape))
fit <- vglm(y ~ 1, genrayleigh, data = rdata, trace = TRUE)
c(with(rdata, mean(y)), head(fitted(fit), 1))
coef(fit, matrix = TRUE)
Coef(fit)
summary(fit)
```

Gensh *Generalized Secant Hyperbolic Distribution*

Description

Density, distribution function, quantile function and random generation for the generalized secant hyperbolic distribution.

Usage

```
dgensh(x, shape, location = 0, scale = 1, tol0 = 1e-4,
       log = FALSE)
pgensh(q, shape, location = 0, scale = 1, tol0 = 1e-4,
       lower.tail = TRUE)
qgensh(p, shape, location = 0, scale = 1, tol0 = 1e-4)
rgensh(n, shape, location = 0, scale = 1, tol0 = 1e-4)
```

Arguments

x, q, p, n, log, lower.tail
: Similar meaning as in Normal.

shape
: Numeric. Shape parameter, called t in Vaughan (2002). Valid values are $-\pi/2 < t$.

location, scale
: Numeric. The location and (positive) scale parameters.

tol0
: Numeric. Used to test whether the shape parameter is close enough to be treated as 0.

Details

This is an implementation of the family of symmetric densities described by Vaughan (2002). By default, the mean and variance are 0 and 1, for all t. Some special (default) cases are: $t = 0$: logistic (which is similar to stats:dt with 9 degrees of freedom); $t = -\pi/2$: the standard secant hyperbolic (whence the name); $t = \infty$: uniform(-sqrt(3), sqrt(3)).

Value

dgensh gives the density, pgensh gives the distribution function, qgensh gives the quantile function, and rgensh generates random deviates.

Warning

Numerical problems may occur when some argument values are extreme.

Author(s)

T. W. Yee.

See Also

gensh, logistic, hypersecant, Logistic.

Examples

```
x <- seq(-2, 4, by = 0.01)
loc <- 1; shape <- -pi /2
## Not run: plot(x, dgensh(x, shape, loc), type = "l",
     main = "Blue is density, orange is the CDF",
     ylim = 0:1, las = 1, ylab = "",
     sub = "Purple are 5, 10, ..., 95 percentiles",
     col = "blue")
abline(h = 0, col = "blue", lty = 2)
lines(qgensh((1:19) / 20, shape, loc), type = "h",
      dgensh(qgensh((1:19) / 20, shape, loc),
             shape, loc), col = "purple", lty = 3)
lines(x, pgensh(x, shape, loc), col = "orange")
abline(h = 0, lty = 2)
## End(Not run)

pp <- (1:19) / 20  # Test two functions
max(abs(pgensh(qgensh(pp, shape, loc),
               shape,loc) - pp))  # Should be 0
```

---

gensh *Generalized Secant Hyperbolic Regression Family Function*

Description

Estimation of the parameters of the generalized secant hyperbolic distribution.

Usage

```
gensh(shape, llocation = "identitylink",
      lscale = "loglink", zero = "scale",
      ilocation = NULL, iscale = NULL, imethod = 1,
      glocation.mux = exp((-4:4)/2),
      gscale.mux = exp((-4:4)/2),
      probs.y = 0.3, tol0 = 1e-4)
```

Arguments

shape — Numeric of length 1. Shape parameter, called t in Vaughan (2002). Valid values are $-\pi/2 < t$.

llocation, lscale — Parameter link functions applied to the two parameters. See Links for more choices. See CommonVGAMffArguments for more information.

zero, imethod — See CommonVGAMffArguments for information.

`ilocation, iscale`
: See `CommonVGAMffArguments` for information.

`glocation.mux, gscale.mux`
: See `CommonVGAMffArguments` for information.

`probs.y, tol0`
: See `CommonVGAMffArguments` for information.

Details

The probability density function of the hyperbolic secant distribution is given by

$$f(y; a, b, s) = [(c_1/b) \ \exp(c_2 z)]/[\exp(2c_2 z) + 2C_3 \exp(c_2 z) + 1]$$

for shape parameter $-\pi < s$ and all real y. The scalars c_1, c_2, C_3 are functions of s. The mean of Y is the location parameter a (returned as the fitted values). All moments of the distribution are finite.

Further details about the parameterization can be found in Vaughan (2002). Fisher scoring is implemented and it has a diagonal EIM. More details are at `Gensh`.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Author(s)

T. W. Yee

References

Vaughan, D. C. (2002). The generalized secant hyperbolic distribution and its properties. *Communications in Statistics—Theory and Methods*, **31**(2): 219–238.

See Also

`hypersecant`, `logistic`.

Examples

```
sh <- -pi / 2; loc <- 2
hdata <- data.frame(x2 = rnorm(nn <- 200))
hdata <- transform(hdata, y = rgensh(nn, sh, loc))
fit <- vglm(y ~ x2, gensh(sh), hdata, trace = TRUE)
coef(fit, matrix = TRUE)
```

`geometric` *Geometric (Truncated and Untruncated) Distributions*

Description

Maximum likelihood estimation for the geometric and truncated geometric distributions.

Usage

```
geometric(link = "logitlink", expected = TRUE, imethod = 1,
          iprob = NULL, zero = NULL)
truncgeometric(upper.limit = Inf,
               link = "logitlink", expected = TRUE, imethod = 1,
               iprob = NULL, zero = NULL)
```

Arguments

`link`	Parameter link function applied to the probability parameter p, which lies in the unit interval. See `Links` for more choices.
`expected`	Logical. Fisher scoring is used if `expected = TRUE`, else Newton-Raphson.
`iprob, imethod, zero`	See `CommonVGAMffArguments` for details.
`upper.limit`	Numeric. Upper values. As a vector, it is recycled across responses first. The default value means both family functions should give the same result.

Details

A random variable Y has a 1-parameter geometric distribution if $P(Y = y) = p(1-p)^y$ for $y = 0, 1, 2, \ldots$. Here, p is the probability of success, and Y is the number of (independent) trials that are fails until a success occurs. Thus the response Y should be a non-negative integer. The mean of Y is $E(Y) = (1-p)/p$ and its variance is $Var(Y) = (1-p)/p^2$. The geometric distribution is a special case of the negative binomial distribution (see `negbinomial`). The geometric distribution is also a special case of the Borel distribution, which is a Lagrangian distribution. If Y has a geometric distribution with parameter p then $Y + 1$ has a positive-geometric distribution with the same parameter. Multiple responses are permitted.

For `truncgeometric()`, the (upper) truncated geometric distribution can have response integer values from 0 to `upper.limit`. It has density `prob * (1 - prob)^y / [1-(1-prob)^(1+upper.limit)]`.

For a generalized truncated geometric distribution with integer values L to U, say, subtract L from the response and feed in $U - L$ as the upper limit.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Author(s)

T. W. Yee. Help from Viet Hoang Quoc is gratefully acknowledged.

References

Forbes, C., Evans, M., Hastings, N. and Peacock, B. (2011). *Statistical Distributions*, Hoboken, NJ, USA: John Wiley and Sons, Fourth edition.

See Also

negbinomial, Geometric, betageometric, expgeometric, zageometric, zigeometric, rbetageom, simulate.vlm.

Examples

```
gdata <- data.frame(x2 = runif(nn <- 1000) - 0.5)
gdata <- transform(gdata, x3 = runif(nn) - 0.5,
                          x4 = runif(nn) - 0.5)
gdata <- transform(gdata, eta  = -1.0 - 1.0 * x2 + 2.0 * x3)
gdata <- transform(gdata, prob = logitlink(eta, inverse = TRUE))
gdata <- transform(gdata, y1 = rgeom(nn, prob))
with(gdata, table(y1))
fit1 <- vglm(y1 ~ x2 + x3 + x4, geometric, data = gdata, trace = TRUE)
coef(fit1, matrix = TRUE)
summary(fit1)

# Truncated geometric (between 0 and upper.limit)
upper.limit <- 5
tdata <- subset(gdata, y1 <= upper.limit)
nrow(tdata)  # Less than nn
fit2 <- vglm(y1 ~ x2 + x3 + x4, truncgeometric(upper.limit),
             data = tdata, trace = TRUE)
coef(fit2, matrix = TRUE)

# Generalized truncated geometric (between lower.limit and upper.limit)
lower.limit <- 1
upper.limit <- 8
gtdata <- subset(gdata, lower.limit <= y1 & y1 <= upper.limit)
with(gtdata, table(y1))
nrow(gtdata)  # Less than nn
fit3 <- vglm(y1 - lower.limit ~ x2 + x3 + x4,
             truncgeometric(upper.limit - lower.limit),
             data = gtdata, trace = TRUE)
coef(fit3, matrix = TRUE)
```

---

get.smart *Retrieve One Component of ".smart.prediction"*

Description

Retrieve one component of the list `.smart.prediction` from `smartpredenv`.

Usage

```
get.smart()
```

Details

`get.smart` is used in `"read"` mode within a smart function: it retrieves parameters saved at the time of fitting, and is used for prediction. `get.smart` is only used in smart functions such as `sm.poly`; `get.smart.prediction` is only used in modelling functions such as `lm` and `glm`. The function `get.smart` gets only a part of `.smart.prediction` whereas `get.smart.prediction` gets the entire `.smart.prediction`.

Value

Returns with one list component of `.smart.prediction` from `smartpredenv`, in fact, `.smart.prediction[[.smart.predi`
The whole procedure mimics a first-in first-out stack (better known as a *queue*).

Side Effects

The variable `.smart.prediction.counter` in `smartpredenv` is incremented beforehand, and then written back to `smartpredenv`.

See Also

`get.smart.prediction`.

Examples

```
print(sm.min1)
```

`get.smart.prediction` *Retrieves ".smart.prediction"*

Description

Retrieves `.smart.prediction` from `smartpredenv`.

Usage

```
get.smart.prediction()
```

Details

A smart modelling function such as `lm` allows smart functions such as `sm.bs` to write to a data structure called `.smart.prediction` in `smartpredenv`. At the end of fitting, `get.smart.prediction` retrieves this data structure. It is then attached to the object, and used for prediction later.

Value

Returns with the list `.smart.prediction` from `smartpredenv`.

See Also

`get.smart`, `lm`.

Examples

```
## Not run:
fit$smart <- get.smart.prediction()  # Put at the end of lm()

## End(Not run)
```

---

gev *Generalized Extreme Value Regression Family Function*

Description

Maximum likelihood estimation of the 3-parameter generalized extreme value (GEV) distribution.

Usage

```
gev(llocation = "identitylink", lscale = "loglink",
    lshape = logofflink(offset = 0.5), percentiles = c(95, 99),
    ilocation = NULL, iscale = NULL, ishape = NULL, imethod = 1,
    gprobs.y = (1:9)/10, gscale.mux = exp((-5:5)/6),
    gshape = (-5:5) / 11 + 0.01,
    iprobs.y = NULL, tolshape0 = 0.001,
    type.fitted = c("percentiles", "mean"),
    zero = c("scale", "shape"))
gevff(llocation = "identitylink", lscale = "loglink",
    lshape = logofflink(offset = 0.5), percentiles = c(95, 99),
    ilocation = NULL, iscale = NULL, ishape = NULL, imethod = 1,
    gprobs.y = (1:9)/10, gscale.mux = exp((-5:5)/6),
    gshape = (-5:5) / 11 + 0.01,
    iprobs.y = NULL, tolshape0 = 0.001,
    type.fitted = c("percentiles", "mean"), zero = c("scale", "shape"))
```

Arguments

`llocation, lscale, lshape`
: Parameter link functions for μ, σ and ξ respectively. See `Links` for more choices.

  For the shape parameter, the default `logofflink` link has an offset called A below; and then the linear/additive predictor is $\log(\xi + A)$ which means that $\xi > -A$. For technical reasons (see **Details**) it is a good idea for $A = 0.5$.

percentiles
: Numeric vector of percentiles used for the fitted values. Values should be between 0 and 100. This argument is ignored if `type.fitted = "mean"`.

type.fitted
: See `CommonVGAMffArguments` for information. The default is to use the `percentiles` argument. If `"mean"` is chosen, then the mean $\mu + \sigma(\Gamma(1-\xi)-1)/\xi$ is returned as the fitted values, and these are only defined for $\xi < 1$.

ilocation, iscale, ishape
: Numeric. Initial value for the location parameter, σ and ξ. A `NULL` means a value is computed internally. The argument `ishape` is more important than the other two. If a failure to converge occurs, or even to obtain initial values occurs, try assigning `ishape` some value (positive or negative; the sign can be very important). Also, in general, a larger value of `iscale` tends to be better than a smaller value.

imethod
: Initialization method. Either the value 1 or 2. If both methods fail then try using `ishape`. See `CommonVGAMffArguments` for information.

gshape
: Numeric vector. The values are used for a grid search for an initial value for ξ. See `CommonVGAMffArguments` for information.

gprobs.y, gscale.mux, iprobs.y
: Numeric vectors, used for the initial values. See `CommonVGAMffArguments` for information.

tolshape0
: Passed into `dgev` when computing the log-likelihood.

zero
: A specifying which linear/additive predictors are modelled as intercepts only. The values can be from the set {1,2,3} corresponding respectively to μ, σ, ξ. If `zero = NULL` then all linear/additive predictors are modelled as a linear combination of the explanatory variables. For many data sets having `zero = 3` is a good idea. See `CommonVGAMffArguments` for information.

Details

The GEV distribution function can be written

$$G(y) = \exp(-[(y-\mu)/\sigma]_+^{-1/\xi})$$

where $\sigma > 0$, $-\infty < \mu < \infty$, and $1 + \xi(y-\mu)/\sigma > 0$. Here, $x_+ = \max(x, 0)$. The μ, σ, ξ are known as the *location*, *scale* and *shape* parameters respectively. The cases $\xi > 0$, $\xi < 0$, $\xi = 0$ correspond to the Frechet, reverse Weibull, and Gumbel types respectively. It can be noted that the Gumbel (or Type I) distribution accommodates many commonly-used distributions such as the normal, lognormal, logistic, gamma, exponential and Weibull.

For the GEV distribution, the kth moment about the mean exists if $\xi < 1/k$. Provided they exist, the mean and variance are given by $\mu + \sigma\{\Gamma(1-\xi) - 1\}/\xi$ and $\sigma^2\{\Gamma(1-2\xi) - \Gamma^2(1-\xi)\}/\xi^2$ respectively, where Γ is the gamma function.

Smith (1985) established that when $\xi > -0.5$, the maximum likelihood estimators are completely regular. To have some control over the estimated ξ try using `lshape = logofflink(offset = 0.5)`, say, or `lshape = extlogitlink(min = -0.5, max = 0.5)`, say.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Warning

Currently, if an estimate of ξ is too close to 0 then an error may occur for `gev()` with multivariate responses. In general, `gevff()` is more reliable than `gev()`.

Fitting the GEV by maximum likelihood estimation can be numerically fraught. If $1+\xi(y-\mu)/\sigma \leq 0$ then some crude evasive action is taken but the estimation process can still fail. This is particularly the case if `vgam` with `s` is used; then smoothing is best done with `vglm` with regression splines (`bs` or `ns`) because `vglm` implements half-stepsizing whereas `vgam` doesn't (half-stepsizing helps handle the problem of straying outside the parameter space.)

Note

The **VGAM** family function `gev` can handle a multivariate (matrix) response, cf. multiple responses. If so, each row of the matrix is sorted into descending order and `NA`s are put last. With a vector or one-column matrix response using `gevff` will give the same result but be faster and it handles the $\xi = 0$ case. The function `gev` implements Tawn (1988) while `gevff` implements Prescott and Walden (1980).

Function `egev()` has been replaced by the new family function `gevff()`. It now conforms to the usual **VGAM** philosophy of having `M1` linear predictors per (independent) response. This is the usual way multiple responses are handled. Hence `vglm(cbind(y1, y2)..., gevff, ...)` will have 6 linear predictors and it is possible to constrain the linear predictors so that the answer is similar to `gev()`. Missing values in the response of `gevff()` will be deleted; this behaviour is the same as with almost every other **VGAM** family function.

The shape parameter ξ is difficult to estimate accurately unless there is a lot of data. Convergence is slow when ξ is near -0.5. Given many explanatory variables, it is often a good idea to make sure `zero = 3`. The range restrictions of the parameter ξ are not enforced; thus it is possible for a violation to occur.

Successful convergence often depends on having a reasonably good initial value for ξ. If failure occurs try various values for the argument `ishape`, and if there are covariates, having `zero = 3` is advised.

Author(s)

T. W. Yee

References

Yee, T. W. and Stephenson, A. G. (2007). Vector generalized linear and additive extreme value models. *Extremes*, **10**, 1–19.

Tawn, J. A. (1988). An extreme-value theory model for dependent observations. *Journal of Hydrology*, **101**, 227–250.

Prescott, P. and Walden, A. T. (1980). Maximum likelihood estimation of the parameters of the generalized extreme-value distribution. *Biometrika*, **67**, 723–724.

Smith, R. L. (1985). Maximum likelihood estimation in a class of nonregular cases. *Biometrika*, **72**, 67–90.

See Also

rgev, gumbel, gumbelff, guplot, rlplot.gevff, gpd, weibullR, frechet, extlogitlink, oxtemp, venice, CommonVGAMffArguments.

Examples

```
## Not run:
# Multivariate example
fit1 <- vgam(cbind(r1, r2) ~ s(year, df = 3), gev(zero = 2:3),
             data = venice, trace = TRUE)
coef(fit1, matrix = TRUE)
head(fitted(fit1))
par(mfrow = c(1, 2), las = 1)
plot(fit1, se = TRUE, lcol = "blue", scol = "forestgreen",
     main = "Fitted mu(year) function (centered)", cex.main = 0.8)
with(venice, matplot(year, depvar(fit1)[, 1:2], ylab = "Sea level (cm)",
     col = 1:2, main = "Highest 2 annual sea levels", cex.main = 0.8))
with(venice, lines(year, fitted(fit1)[,1], lty = "dashed", col = "blue"))
legend("topleft", lty = "dashed", col = "blue", "Fitted 95 percentile")

# Univariate example
(fit <- vglm(maxtemp ~ 1, gevff, data = oxtemp, trace = TRUE))
head(fitted(fit))
coef(fit, matrix = TRUE)
Coef(fit)
vcov(fit)
vcov(fit, untransform = TRUE)
sqrt(diag(vcov(fit)))  # Approximate standard errors
rlplot(fit)

## End(Not run)
```

---

gevUC *The Generalized Extreme Value Distribution*

Description

Density, distribution function, quantile function and random generation for the generalized extreme value distribution (GEV) with location parameter `location`, scale parameter `scale` and shape parameter shape.

Usage

```
dgev(x, location = 0, scale = 1, shape = 0, log = FALSE,
     tolshape0 = sqrt(.Machine$double.eps))
pgev(q, location = 0, scale = 1, shape = 0, lower.tail = TRUE, log.p = FALSE)
qgev(p, location = 0, scale = 1, shape = 0, lower.tail = TRUE, log.p = FALSE)
rgev(n, location = 0, scale = 1, shape = 0)
```

Arguments

- `x, q` — vector of quantiles.
- `p` — vector of probabilities.
- `n` — number of observations. If `length(n) > 1` then the length is taken to be the number required.
- `location` — the location parameter μ.
- `scale` — the (positive) scale parameter σ. Must consist of positive values.
- `shape` — the shape parameter ξ.
- `log` — Logical. If `log = TRUE` then the logarithm of the density is returned.
- `lower.tail, log.p` — Same meaning as in `punif` or `qunif`.
- `tolshape0` — Positive numeric. Threshold/tolerance value for resting whether ξ is zero. If the absolute value of the estimate of ξ is less than this value then it will be assumed zero and a Gumbel distribution will be used.

Details

See `gev`, the **VGAM** family function for estimating the 3 parameters by maximum likelihood estimation, for formulae and other details. Apart from `n`, all the above arguments may be vectors and are recyled to the appropriate length if necessary.

Value

`dgev` gives the density, `pgev` gives the distribution function, `qgev` gives the quantile function, and `rgev` generates random deviates.

Note

The default value of $\xi = 0$ means the default distribution is the Gumbel.

Currently, these functions have different argument names compared with those in the **evd** package.

Author(s)

T. W. Yee

References

Coles, S. (2001). *An Introduction to Statistical Modeling of Extreme Values*. London: Springer-Verlag.

See Also

`gev`, `gevff`, `vglm.control`.

Examples

```
 loc <- 2; sigma <- 1; xi <- -0.4
pgev(qgev(seq(0.05, 0.95, by = 0.05), loc, sigma, xi), loc, sigma, xi)
## Not run:  x <- seq(loc - 3, loc + 3, by = 0.01)
plot(x, dgev(x, loc, sigma, xi), type = "l", col = "blue", ylim = c(0, 1),
     main = "Blue is density, orange is the CDF",
     sub = "Purple are 10,...,90 percentiles", ylab = "", las = 1)
abline(h = 0, col = "blue", lty = 2)
lines(qgev(seq(0.1, 0.9, by = 0.1), loc, sigma, xi),
      dgev(qgev(seq(0.1, 0.9, by = 0.1), loc, sigma, xi), loc, sigma, xi),
      col = "purple", lty = 3, type = "h")
lines(x, pgev(x, loc, sigma, xi), type = "l", col = "orange")
abline(h = (0:10)/10, lty = 2, col = "gray50")

## End(Not run)
```

---

gew *General Electric and Westinghouse Data*

Description

General Electric and Westinghouse capital data.

Usage

```
data(gew)
```

Format

A data frame with 20 observations on the following 7 variables. All variables are numeric vectors. Variables ending in `.g` correspond to General Electric and those ending in `.w` are Westinghouse.

year The observations are the years from 1934 to 1953

invest.g, invest.w investment figures. These are I = Gross investment = additions to plant and equipment plus maintenance and repairs in millions of dollars deflated by P_1.

capital.g, capital.w capital stocks. These are C = The stock of plant and equipment = accumulated sum of net additions to plant and equipment deflated by P_1 minus depreciation allowance deflated by P_3.

value.g, value.w market values. These are F = Value of the firm = price of common and preferred shares at December 31 (or average price of December 31 and January 31 of the following year) times number of common and preferred shares outstanding plus total book value of debt at December 31 in millions of dollars deflated by P_2.

Details

These data are a subset of a table in Boot and de Wit (1960), also known as the Grunfeld data. It is used a lot in econometrics, e.g., for seemingly unrelated regressions (see `SURff`).

Here, $P_1 =$ Implicit price deflator of producers durable equipment (base 1947), $P_2 =$ Implicit price deflator of G.N.P. (base 1947), $P_3 =$ Depreciation expense deflator = ten years moving average of wholesale price index of metals and metal products (base 1947).

Source

Table 10 of: Boot, J. C. G. and de Wit, G. M. (1960) Investment Demand: An Empirical Contribution to the Aggregation Problem. *International Economic Review*, **1**, 3–30.

Grunfeld, Y. (1958) The Determinants of Corporate Investment. Unpublished PhD Thesis (Chicago).

References

Zellner, A. (1962). An efficient method of estimating seemingly unrelated regressions and tests for aggregation bias. *Journal of the American Statistical Association*, **57**, 348–368.

See Also

`SURff`, `http://statmath.wu.ac.at/~zeileis/grunfeld` (the link might now be stale).

Examples

```
str(gew)
```

`goffset` *GAITD Offset for the GTE Method*

Description

Utility function to create a matrix of log-offset values, to help facilitate the Generally-Truncated-Expansion method

Usage

```
goffset(mux, n,
        a.mix = NULL, i.mix = NULL, d.mix = NULL,
        a.mlm = NULL, i.mlm = NULL, d.mlm = NULL, par1or2 = 1)
```

Arguments

- `mux` — Multiplier. Usually a small positive integer. Must be positive. The value 1 means no change.
- `n` — Number of rows. A positive integer, it should be the number of rows of the data frame containing the data.
- `a.mix, i.mix, d.mix` — See, e.g., `gaitdpoisson`.
- `a.mlm, i.mlm, d.mlm` — See, e.g., `gaitdpoisson`.
- `par1or2` — Number of parameters of the parent distribution. Set `par1or2 = 2` for `gaitdnbinomial`, else the default value should be used.

Details

This function is intended to make the Generally-Truncated-Expansion (GTE) method easier for the user. It only makes sense if the linear predictors(s) are log of the mean of the parent distribution, which is the usual case for `gaitdpoisson` and `gaitdnbinomial`. However, for `gaitdlog` and `gaitdzeta` one should be using `logffMlink` and `zetaffMlink`.

Without this function, the user must do quite a lot of book-keeping to know which columns of the offset matrix is to be assigned `log(mux)`. This can be rather laborious.

In the fictitional example below the response is underdispersed with respect to a Poisson distribution and doubling the response achieves approximate equidispersion.

Value

A matrix with `n` rows and the same number of columns that a GAITD regression would produce for its matrix of linear predictors. The matrix can be inputted into `vglm` by assigning the `offset` argument.

Note

This function is still in a developmental stage. The order of the arguments might change, hence it's safest to invoke it with full specification.

See Also

`gaitdpoisson`, `gaitdlog`, `gaitdzeta`, `gaitdnbinomial`, `Trunc`, `offset`.

Examples

```
i.mix <- c(5, 10, 15, 20); a.mlm <- 13; mymux <- 2
goffset(mymux, 10, i.mix = i.mix, a.mlm = a.mlm)
## Not run: org1  <- with(gdata, range(y))  # Original range of the data
vglm(mymux * y ~ 1,
     offset = goffset(mymux, nrow(gdata), i.mix = i.mix, a.mlm = a.mlm),
     gaitdpoisson(a.mlm = mymux * a.mlm, i.mix = mymux * i.mix,
                  truncate = Trunc(org1, mymux)),
     data = gdata)
```

```
## End(Not run)
```

Gompertz *Gompertz Distribution*

Description

Density, cumulative distribution function, quantile function and random generation for the Gompertz distribution.

Usage

```
dgompertz(x, scale = 1, shape, log = FALSE)
pgompertz(q, scale = 1, shape, lower.tail = TRUE, log.p = FALSE)
qgompertz(p, scale = 1, shape, lower.tail = TRUE, log.p = FALSE)
rgompertz(n, scale = 1, shape)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. Same as in `runif`.
`log`	Logical. If `log = TRUE` then the logarithm of the density is returned.
`lower.tail, log.p`	Same meaning as in `pnorm` or `qnorm`.
`scale, shape`	positive scale and shape parameters.

Details

See `gompertz` for details.

Value

`dgompertz` gives the density, `pgompertz` gives the cumulative distribution function, `qgompertz` gives the quantile function, and `rgompertz` generates random deviates.

Author(s)

T. W. Yee and Kai Huang

See Also

`gompertz`, `dgumbel`, `dmakeham`.

Examples

```
probs <- seq(0.01, 0.99, by = 0.01)
Shape <- exp(1); Scale <- exp(1)
max(abs(pgompertz(qgompertz(p = probs, Scale, shape = Shape),
                  Scale, shape = Shape) - probs))  # Should be 0

## Not run:  x <- seq(-0.1, 1.0, by = 0.001)
plot(x, dgompertz(x, Scale,shape = Shape), type = "l", las = 1,
     main = "Blue is density, orange is the CDF", col = "blue",
     sub = "Purple lines are the 10,20,...,90 percentiles",
     ylab = "")
abline(h = 0, col = "blue", lty = 2)
lines(x, pgompertz(x, Scale, shape = Shape), col = "orange")
probs <- seq(0.1, 0.9, by = 0.1)
Q <- qgompertz(probs, Scale, shape = Shape)
lines(Q, dgompertz(Q, Scale, shape = Shape), col = "purple",
      lty = 3, type = "h")
pgompertz(Q, Scale, shape = Shape) - probs  # Should be all zero
abline(h = probs, col = "purple", lty = 3)
## End(Not run)
```

`gompertz` *Gompertz Regression Family Function*

Description

Maximum likelihood estimation of the 2-parameter Gompertz distribution.

Usage

```
gompertz(lscale = "loglink", lshape = "loglink",
         iscale = NULL,   ishape = NULL,
         nsimEIM = 500, zero = NULL, nowarning = FALSE)
```

Arguments

`nowarning`	Logical. Suppress a warning? Ignored for **VGAM** 0.9-7 and higher.
`lshape, lscale`	Parameter link functions applied to the shape parameter `a`, scale parameter `scale`. All parameters are positive. See `Links` for more choices.
`ishape, iscale`	Optional initial values. A `NULL` means a value is computed internally.
`nsimEIM, zero`	See `CommonVGAMffArguments`.

Details

The Gompertz distribution has a cumulative distribution function

$$F(x; \alpha, \beta) = 1 - \exp[-(\alpha/\beta) \times (\exp(\beta x) - 1)]$$

which leads to a probability density function

$$f(x; \alpha, \beta) = \alpha \exp(\beta x) \exp[-(\alpha/\beta) \times (\exp(\beta x) - 1)]$$

for $\alpha > 0$, $\beta > 0$, $x > 0$. Here, β is called the scale parameter `scale`, and α is called the shape parameter (one could refer to α as a location parameter and β as a shape parameter—see Lenart (2014)). The mean is involves an exponential integral function. Simulated Fisher scoring is used and multiple responses are handled.

The Makeham distibution has an additional parameter compared to the Gompertz distribution. If X is defined to be the result of sampling from a Gumbel distribution until a negative value Z is produced, then $X = -Z$ has a Gompertz distribution.

Value

An object of class "vglmff" (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Warning

The same warnings in `makeham` apply here too.

Author(s)

T. W. Yee

References

Lenart, A. (2014). The moments of the Gompertz distribution and maximum likelihood estimation of its parameters. *Scandinavian Actuarial Journal*, **2014**, 255–277.

See Also

`dgompertz`, `makeham`, `simulate.vlm`.

Examples

```
## Not run:
gdata <- data.frame(x2 = runif(nn <- 1000))
gdata <- transform(gdata, eta1  = -1,
                          eta2  = -1 + 0.2 * x2,
                          ceta1 =  1,
                          ceta2 = -1 + 0.2 * x2)
gdata <- transform(gdata, shape1 = exp(eta1),
                          shape2 = exp(eta2),
                          scale1 = exp(ceta1),
                          scale2 = exp(ceta2))
gdata <- transform(gdata, y1 = rgompertz(nn, scale = scale1, shape = shape1),
                          y2 = rgompertz(nn, scale = scale2, shape = shape2))

fit1 <- vglm(y1 ~ 1,  gompertz, data = gdata, trace = TRUE)
fit2 <- vglm(y2 ~ x2, gompertz, data = gdata, trace = TRUE)
```

```
coef(fit1, matrix = TRUE)
Coef(fit1)
summary(fit1)
coef(fit2, matrix = TRUE)
summary(fit2)

## End(Not run)
```

---

gpd *Generalized Pareto Distribution Regression Family Function*

Description

Maximum likelihood estimation of the 2-parameter generalized Pareto distribution (GPD).

Usage

```
gpd(threshold = 0, lscale = "loglink", lshape = logofflink(offset = 0.5),
    percentiles = c(90, 95), iscale = NULL, ishape = NULL,
    tolshape0 = 0.001, type.fitted = c("percentiles", "mean"),
    imethod = 1, zero = "shape")
```

Arguments

`threshold` Numeric, values are recycled if necessary. The threshold value(s), called μ below.

`lscale` Parameter link function for the scale parameter σ. See `Links` for more choices.

`lshape` Parameter link function for the shape parameter ξ. See `Links` for more choices. The default constrains the parameter to be greater than -0.5 because if $\xi \leq -0.5$ then Fisher scoring does not work. See the Details section below for more information.

For the shape parameter, the default `logofflink` link has an offset called A below; and then the second linear/additive predictor is $\log(\xi + A)$ which means that $\xi > -A$. The working weight matrices are positive definite if $A = 0.5$.

`percentiles` Numeric vector of percentiles used for the fitted values. Values should be between 0 and 100. See the example below for illustration. This argument is ignored if `type.fitted = "mean"`.

`type.fitted` See `CommonVGAMffArguments` for information. The default is to use the `percentiles` argument. If `"mean"` is chosen, then the mean $\mu + \sigma/(1 - \xi)$ is returned as the fitted values, and these are only defined for $\xi < 1$.

`iscale, ishape` Numeric. Optional initial values for σ and ξ. The default is to use `imethod` and compute a value internally for each parameter. Values of `ishape` should be between -0.5 and 1. Values of `iscale` should be positive.

`tolshape0` Passed into `dgpd` when computing the log-likelihood.

`imethod` Method of initialization, either 1 or 2. The first is the method of moments, and the second is a variant of this. If neither work, try assigning values to arguments `ishape` and/or `iscale`.

`zero` Can be an integer-valued vector specifying which linear/additive predictors are modelled as intercepts only. For one response, the value should be from the set {1,2} corresponding respectively to σ and ξ. It is often a good idea for the σ parameter only to be modelled through a linear combination of the explanatory variables because the shape parameter is probably best left as an intercept only: `zero = 2`. Setting `zero = NULL` means both parameters are modelled with explanatory variables. See `CommonVGAMffArguments` for more details.

Details

The distribution function of the GPD can be written

$$G(y) = 1 - [1 + \xi(y - \mu)/\sigma]_{+}^{-1/\xi}$$

where μ is the location parameter (known, with value `threshold`), $\sigma > 0$ is the scale parameter, ξ is the shape parameter, and $h_{+} = \max(h, 0)$. The function $1 - G$ is known as the *survivor function*. The limit $\xi \to 0$ gives the *shifted exponential* as a special case:

$$G(y) = 1 - \exp[-(y - \mu)/\sigma].$$

The support is $y > \mu$ for $\xi > 0$, and $\mu < y < \mu - \sigma/\xi$ for $\xi < 0$.

Smith (1985) showed that if $\xi <= -0.5$ then this is known as the nonregular case and problems/difficulties can arise both theoretically and numerically. For the (regular) case $\xi > -0.5$ the classical asymptotic theory of maximum likelihood estimators is applicable; this is the default.

Although for $\xi < -0.5$ the usual asymptotic properties do not apply, the maximum likelihood estimator generally exists and is superefficient for $-1 < \xi < -0.5$, so it is “better” than normal. When $\xi < -1$ the maximum likelihood estimator generally does not exist as it effectively becomes a two parameter problem.

The mean of Y does not exist unless $\xi < 1$, and the variance does not exist unless $\xi < 0.5$. So if you want to fit a model with finite variance use `lshape = "extlogitlink"`.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`. However, for this **VGAM** family function, `vglm` is probably preferred over `vgam` when there is smoothing.

Warning

Fitting the GPD by maximum likelihood estimation can be numerically fraught. If $1 + \xi(y - \mu)/\sigma \leq 0$ then some crude evasive action is taken but the estimation process can still fail. This is particularly the case if `vgam` with `s` is used. Then smoothing is best done with `vglm` with regression splines (`bs` or `ns`) because `vglm` implements half-stepsizing whereas `vgam` doesn't. Half-stepsizing helps handle the problem of straying outside the parameter space.

Note

The response in the formula of vglm and vgam is y. Internally, $y - \mu$ is computed. This **VGAM** family function can handle a multiple responses, which is inputted as a matrix. The response stored on the object is the original uncentred data.

With functions rgpd, dgpd, etc., the argument location matches with the argument threshold here.

Author(s)

T. W. Yee

References

Yee, T. W. and Stephenson, A. G. (2007). Vector generalized linear and additive extreme value models. *Extremes*, **10**, 1–19.

Coles, S. (2001). *An Introduction to Statistical Modeling of Extreme Values*. London: Springer-Verlag.

Smith, R. L. (1985). Maximum likelihood estimation in a class of nonregular cases. *Biometrika*, **72**, 67–90.

See Also

rgpd, meplot, gev, paretoff, vglm, vgam, s.

Examples

```
# Simulated data from an exponential distribution (xi = 0)
Threshold <- 0.5
gdata <- data.frame(y1 = Threshold + rexp(n = 3000, rate = 2))
fit <- vglm(y1 ~ 1, gpd(threshold = Threshold), data = gdata, trace = TRUE)
head(fitted(fit))
summary(depvar(fit))  # The original uncentred data
coef(fit, matrix = TRUE)  # xi should be close to 0
Coef(fit)
summary(fit)

head(fit@extra$threshold)  # Note the threshold is stored here

# Check the 90 percentile
ii <- depvar(fit) < fitted(fit)[1, "90%"]
100 * table(ii) / sum(table(ii))  # Should be 90%

# Check the 95 percentile
ii <- depvar(fit) < fitted(fit)[1, "95%"]
100 * table(ii) / sum(table(ii))  # Should be 95%

## Not run:  plot(depvar(fit), col = "blue", las = 1,
               main = "Fitted 90% and 95% quantiles")
matlines(1:length(depvar(fit)), fitted(fit), lty = 2:3, lwd = 2)
## End(Not run)
```

```
# Another example
gdata <- data.frame(x2 = runif(nn <- 2000))
Threshold <- 0; xi <- exp(-0.8) - 0.5
gdata <- transform(gdata, y2 = rgpd(nn, scale = exp(1 + 0.1*x2), shape = xi))
fit <- vglm(y2 ~ x2, gpd(Threshold), data = gdata, trace = TRUE)
coef(fit, matrix = TRUE)


## Not run:  # Nonparametric fits
# Not so recommended:
fit1 <- vgam(y2 ~ s(x2), gpd(Threshold), data = gdata, trace = TRUE)
par(mfrow = c(2, 1))
plot(fit1, se = TRUE, scol = "blue")
# More recommended:
fit2 <- vglm(y2 ~ sm.bs(x2), gpd(Threshold), data = gdata, trace = TRUE)
plot(as(fit2, "vgam"), se = TRUE, scol = "blue")
## End(Not run)
```

gpdUC *The Generalized Pareto Distribution*

Description

Density, distribution function, quantile function and random generation for the generalized Pareto distribution (GPD) with location parameter `location`, scale parameter `scale` and shape parameter `shape`.

Usage

```
dgpd(x, location = 0, scale = 1, shape = 0, log = FALSE,
     tolshape0 = sqrt(.Machine$double.eps))
pgpd(q, location = 0, scale = 1, shape = 0,
     lower.tail = TRUE, log.p = FALSE)
qgpd(p, location = 0, scale = 1, shape = 0,
     lower.tail = TRUE, log.p = FALSE)
rgpd(n, location = 0, scale = 1, shape = 0)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. If `length(n) > 1` then the length is taken to be the number required.
`location`	the location parameter μ.
`scale`	the (positive) scale parameter σ.

`shape` the shape parameter ξ.

`log` Logical. If `log = TRUE` then the logarithm of the density is returned.

`lower.tail, log.p`
Same meaning as in `punif` or `qunif`.

`tolshape0` Positive numeric. Threshold/tolerance value for resting whether ξ is zero. If the absolute value of the estimate of ξ is less than this value then it will be assumed zero and an exponential distribution will be used.

Details

See `gpd`, the **VGAM** family function for estimating the two parameters by maximum likelihood estimation, for formulae and other details. Apart from `n`, all the above arguments may be vectors and are recyled to the appropriate length if necessary.

Value

`dgpd` gives the density, `pgpd` gives the distribution function, `qgpd` gives the quantile function, and `rgpd` generates random deviates.

Note

The default values of all three parameters, especially $\xi = 0$, means the default distribution is the exponential.

Currently, these functions have different argument names compared with those in the **evd** package.

Author(s)

T. W. Yee and Kai Huang

References

Coles, S. (2001). *An Introduction to Statistical Modeling of Extreme Values*. London: Springer-Verlag.

See Also

`gpd`, `Exponential`.

Examples

```
## Not run:  loc <- 2; sigma <- 1; xi <- -0.4
x <- seq(loc - 0.2, loc + 3, by = 0.01)
plot(x, dgpd(x, loc, sigma, xi), type = "l", col = "blue",
     main = "Blue is density, red is the CDF", ylim = c(0, 1),
     sub = "Purple are 5,10,...,95 percentiles", ylab = "", las = 1)
abline(h = 0, col = "blue", lty = 2)
lines(qgpd(seq(0.05, 0.95, by = 0.05), loc, sigma, xi),
  dgpd(qgpd(seq(0.05, 0.95, by = 0.05), loc, sigma, xi), loc, sigma, xi),
      col = "purple", lty = 3, type = "h")
lines(x, pgpd(x, loc, sigma, xi), type = "l", col = "red")
```

```
abline(h = 0, lty = 2)

pgpd(qgpd(seq(0.05, 0.95, by = 0.05), loc, sigma, xi), loc, sigma, xi)

## End(Not run)
```

---

grain.us *Grain Prices Data in USA*

Description

A 4-column matrix.

Usage

```
data(grain.us)
```

Format

The columns are:

wheat.flour numeric

corn numeric

wheat numeric

rye numeric

Details

Monthly averages of grain prices in the United States for wheat flour, corn, wheat, and rye for the period January 1961 through October 1972. The units are US dollars per 100 pound sack for wheat flour, and per bushel for corn, wheat and rye.

Source

Ahn and Reinsel (1988).

References

Ahn, S. K and Reinsel, G. C. (1988). Nested reduced-rank autoregressive models for multiple time series. *Journal of the American Statistical Association*, **83**, 849–856.

Examples

```
## Not run:
cgrain <- scale(grain.us, scale = FALSE)  # Center the time series only
fit <- vglm(cgrain ~ 1, rrar(Rank = c(4, 1)),
            epsilon = 1e-3, stepsize = 0.5, trace = TRUE, maxit = 50)
summary(fit)

## End(Not run)
```

---

grc *Row-Column Interaction Models including Goodman's RC Association Model and Unconstrained Quadratic Ordination*

Description

Fits a Goodman's RC association model (GRC) to a matrix of counts, and more generally, row-column interaction models (RCIMs). RCIMs allow for unconstrained quadratic ordination (UQO).

Usage

```
grc(y, Rank = 1, Index.corner = 2:(1 + Rank),
    str0 = 1, summary.arg = FALSE, h.step = 1e-04, ...)
rcim(y, family = poissonff, Rank = 0, M1 = NULL,
     weights = NULL, which.linpred = 1,
     Index.corner = ifelse(is.null(str0), 0, max(str0)) + 1:Rank,
     rprefix = "Row.", cprefix = "Col.", iprefix = "X2.",
     offset = 0, str0 = if (Rank) 1 else NULL,
     summary.arg = FALSE, h.step = 0.0001,
     rbaseline = 1, cbaseline = 1,
     has.intercept = TRUE, M = NULL,
     rindex = 2:nrow(y), cindex = 2:ncol(y), iindex = 2:nrow(y), ...)
```

Arguments

`y` For `grc()`: a matrix of counts. For `rcim()`: a general matrix response depending on `family`. Output from `table()` is acceptable; it is converted into a matrix. Note that `y` should be at least 3 by 3 in dimension.

`family` A **VGAM** family function. By default, the first linear/additive predictor is fitted using main effects plus an optional rank-`Rank` interaction term. Not all family functions are suitable or make sense. All other linear/additive predictors are fitted using an intercept-only, so it has a common value over all rows and columns. For example, `zipoissonff` may be suitable for counts but not `zipoisson` because of the ordering of the linear/additive predictors. If the **VGAM** family function does not have an `infos` slot then `M1` needs to be inputted (the number of linear predictors for an ordinary (usually univariate) response, aka M). The **VGAM** family function also needs to be able to handle multiple responses (currently not all of them can do this).

`Rank` An integer from the set {0,...,`min(nrow(y), ncol(y))`}. This is the dimension of the fit in terms of the interaction. For `grc()` this argument must be positive. A value of 0 means no interactions (i.e., main effects only); each row and column is represented by an indicator variable.

`weights` Prior weights. Fed into `rrvglm` or `vglm`.

`which.linpred` Single integer. Specifies which linear predictor is modelled as the sum of an intercept, row effect, column effect plus an optional interaction term. It should be one value from the set `1:M1`.

Index.corner
: A vector of Rank integers. These are used to store the Rank by Rank identity matrix in the A matrix; corner constraints are used.

rprefix, cprefix, iprefix
: Character, for rows and columns and interactions respectively. For labelling the indicator variables.

offset
: Numeric. Either a matrix of the right dimension, else a single numeric expanded into such a matrix.

str0
: Ignored if Rank = 0, else an integer from the set {1,...,min(nrow(y), ncol(y))}, specifying the row that is used as the structural zero. Passed into rrvglm.control if Rank > 0. Set str0 = NULL for none.

summary.arg
: Logical. If TRUE then a summary is returned. If TRUE then y may be the output (fitted object) of grc().

h.step
: A small positive value that is passed into summary.rrvglm(). Only used when summary.arg = TRUE.

...
: Arguments that are passed into rrvglm.control().

M1
: The number of linear predictors of the **VGAM** family function for an ordinary (univariate) response. Then the number of linear predictors of the rcim() fit is usually the number of columns of y multiplied by M1. The default is to evaluate the infos slot of the **VGAM** family function to try to evaluate it; see vglmff-class. If this information is not yet supplied by the family function then the value needs to be inputted manually using this argument.

rbaseline, cbaseline
: Baseline reference levels for the rows and columns. Currently stored on the object but not used.

has.intercept
: Logical. Include an intercept?

M, cindex
: M is the usual **VGAM** M, viz. the number of linear/additive predictors in total. Also, cindex means column index, and these point to the columns of y which are part of the vector of linear/additive predictor *main effects*.

  For family = multinomial it is necessary to input these arguments as M = ncol(y)-1 and cindex = 2:(ncol(y)-1).

rindex, iindex
: rindex means row index, and these are similar to cindex. iindex means interaction index, and these are similar to cindex.

Details

Goodman's RC association model fits a reduced-rank approximation to a table of counts. A Poisson model is assumed. The log of each cell mean is decomposed as an intercept plus a row effect plus a column effect plus a reduced-rank component. The latter can be collectively written A %*% t(C), the product of two 'thin' matrices. Indeed, A and C have Rank columns. By default, the first column and row of the interaction matrix A %*% t(C) is chosen to be structural zeros, because str0 = 1. This means the first row of A are all zeros.

This function uses options()$contrasts to set up the row and column indicator variables. In particular, Equation (4.5) of Yee and Hastie (2003) is used. These are called Row. and Col. (by default) followed by the row or column number.

The function rcim() is more general than grc(). Its default is a no-interaction model of grc(), i.e., rank-0 and a Poisson distribution. This means that each row and column has a dummy variable associated with it. The first row and first column are baseline. The power of rcim() is that many **VGAM** family functions can be assigned to its family argument. For example, uninormal fits something in between a 2-way ANOVA with and without interactions, alaplace2 with Rank = 0 is something like medpolish. Others include zipoissonff and negbinomial. Hopefully one day *all* **VGAM** family functions will work when assigned to the family argument, although the result may not have meaning.

Unconstrained quadratic ordination (UQO) can be performed using rcim() and grc(). This has been called *unconstrained Gaussian ordination* in the literature, however the word *Gaussian* has two meanings which is confusing; it is better to use *quadratic* because the bell-shape response surface is meant. UQO is similar to CQO (cqo) except there are no environmental/explanatory variables. Here, a GLM is fitted to each column (species) that is a quadratic function of hypothetical latent variables or gradients. Thus each row of the response has an associated site score, and each column of the response has an associated optimum and tolerance matrix. UQO can be performed here under the assumption that all species have the same tolerance matrices. See Yee and Hadi (2014) for details. It is not recommended that presence/absence data be inputted because the information content is so low for each site-species cell. The example below uses Poisson counts.

Value

An object of class "grc", which currently is the same as an "rrvglm" object. Currently, a rank-0 rcim() object is of class rcim0-class, else of class "rcim" (this may change in the future).

Warning

The function rcim() is experimental at this stage and may have bugs. Quite a lot of expertise is needed when fitting and in its interpretion thereof. For example, the constraint matrices applies the reduced-rank regression to the first (see which.linpred) linear predictor and the other linear predictors are intercept-only and have a common value throughout the entire data set. This means that, by default, family = zipoissonff is appropriate but not family = zipoisson. Else set family = zipoisson and which.linpred = 2. To understand what is going on, do examine the constraint matrices of the fitted object, and reconcile this with Equations (4.3) to (4.5) of Yee and Hastie (2003).

The functions temporarily create a permanent data frame called .grc.df or .rcim.df, which used to be needed by summary.rrvglm(). Then these data frames are deleted before exiting the function. If an error occurs then the data frames may be present in the workspace.

Note

These functions set up the indicator variables etc. before calling rrvglm or vglm. The ... is passed into rrvglm.control or vglm.control, This means, e.g., Rank = 1 is default for grc().

The data should be labelled with rownames and colnames. Setting trace = TRUE is recommended to monitor convergence. Using criterion = "coefficients" can result in slow convergence.

If summary = TRUE then y can be a "grc" object, in which case a summary can be returned. That is, grc(y, summary = TRUE) is equivalent to summary(grc(y)). It is not possible to plot a grc(y, summary = TRUE) or rcim(y, summary = TRUE) object.

Author(s)

Thomas W. Yee, with assistance from Alfian F. Hadi.

References

Yee, T. W. and Hastie, T. J. (2003). Reduced-rank vector generalized linear models. *Statistical Modelling*, **3**, 15–41.

Yee, T. W. and Hadi, A. F. (2014). Row-column interaction models, with an R implementation. *Computational Statistics*, **29**, 1427–1445.

Goodman, L. A. (1981). Association models and canonical correlation in the analysis of cross-classifications having ordered categories. *Journal of the American Statistical Association*, **76**, 320–334.

See Also

rrvglm, rrvglm.control, rrvglm-class, summary.grc, moffset, Rcim, Select, Qvar, plotrcim0, cqo, multinomial, alcoff, crashi, auuc, olym08, olym12, poissonff, medpolish.

Examples

```
# Example 1: Undergraduate enrolments at Auckland University in 1990
fitted(grc1 <- grc(auuc))
summary(grc1)

grc2 <- grc(auuc, Rank = 2, Index.corner = c(2, 5))
fitted(grc2)
summary(grc2)

model3 <- rcim(auuc, Rank = 1, fam = multinomial,
               M = ncol(auuc)-1, cindex = 2:(ncol(auuc)-1), trace = TRUE)
fitted(model3)
summary(model3)

# Median polish but not 100 percent reliable. Maybe call alaplace2()...
## Not run:
rcim0 <- rcim(auuc, fam = alaplace1(tau = 0.5), trace=FALSE, maxit = 500)
round(fitted(rcim0), digits = 0)
round(100 * (fitted(rcim0) - auuc) / auuc, digits = 0)  # Discrepancy
depvar(rcim0)
round(coef(rcim0, matrix = TRUE), digits = 2)
Coef(rcim0, matrix = TRUE)
# constraints(rcim0)
names(constraints(rcim0))

# Compare with medpolish():
(med.a <- medpolish(auuc))
fv <- med.a$overall + outer(med.a$row, med.a$col, "+")
round(100 * (fitted(rcim0) - fv) / fv)  # Hopefully should be all 0s

## End(Not run)
```

```
# Example 2: 2012 Summer Olympic Games in London
## Not run:  top10 <- head(olym12, 10)
grc1.oly12 <- with(top10, grc(cbind(gold, silver, bronze)))
round(fitted(grc1.oly12))
round(resid(grc1.oly12, type = "response"), digits = 1)  # Resp. resids
summary(grc1.oly12)
Coef(grc1.oly12)

## End(Not run)


# Example 3: UQO; see Yee and Hadi (2014)
## Not run:
n <- 100; p <- 5; S <- 10
pdata <- rcqo(n, p, S, es.opt = FALSE, eq.max = FALSE,
              eq.tol = TRUE, sd.latvar = 0.75)  # Poisson counts
true.nu <- attr(pdata, "latvar")  # The 'truth'; site scores
attr(pdata, "tolerances")  # The 'truth'; tolerances

Y <- Select(pdata, "y", sort = FALSE)  # Y matrix (n x S); the "y" vars
uqo.rcim1 <- rcim(Y, Rank = 1,
                  str0 = NULL,  # Delta covers entire n x M matrix
                  iindex = 1:nrow(Y),  # RRR covers the entire Y
                  has.intercept = FALSE)  # Suppress the intercept

# Plot 1
par(mfrow = c(2, 2))
plot(attr(pdata, "optimums"), Coef(uqo.rcim1)@A,
     col = "blue", type = "p", main = "(a) UQO optimums",
     xlab = "True optimums", ylab = "Estimated (UQO) optimums")
mylm <- lm(Coef(uqo.rcim1)@A ~ attr(pdata, "optimums"))
abline(coef = coef(mylm), col = "orange", lty = "dashed")

# Plot 2
fill.val <- NULL  # Choose this for the new parameterization
plot(attr(pdata, "latvar"), c(fill.val, concoef(uqo.rcim1)),
     las = 1, col = "blue", type = "p", main = "(b) UQO site scores",
     xlab = "True site scores", ylab = "Estimated (UQO) site scores" )
mylm <- lm(c(fill.val, concoef(uqo.rcim1)) ~ attr(pdata, "latvar"))
abline(coef = coef(mylm), col = "orange", lty = "dashed")

# Plots 3 and 4
myform <- attr(pdata, "formula")
p1ut <- cqo(myform, family = poissonff,
            eq.tol = FALSE, trace = FALSE, data = pdata)
c1ut <- cqo(Select(pdata, "y", sort = FALSE) ~ scale(latvar(uqo.rcim1)),
        family = poissonff, eq.tol = FALSE, trace = FALSE, data = pdata)
lvplot(p1ut, lcol = 1:S, y = TRUE, pcol = 1:S, pch = 1:S, pcex = 0.5,
       main = "(c) CQO fitted to the original data",
       xlab = "Estimated (CQO) site scores")
lvplot(c1ut, lcol = 1:S, y = TRUE, pcol = 1:S, pch = 1:S, pcex = 0.5,
```

```
main = "(d) CQO fitted to the scaled UQO site scores",
xlab = "Estimated (UQO) site scores")

## End(Not run)
```

gumbel *Gumbel Regression Family Function*

Description

Maximum likelihood estimation of the 2-parameter Gumbel distribution.

Usage

```
gumbel(llocation = "identitylink", lscale = "loglink",
       iscale = NULL, R = NA, percentiles = c(95, 99),
       mpv = FALSE, zero = NULL)
gumbelff(llocation = "identitylink", lscale = "loglink",
         iscale = NULL, R = NA, percentiles = c(95, 99),
         zero = "scale", mpv = FALSE)
```

Arguments

`llocation, lscale`
: Parameter link functions for μ and σ. See `Links` for more choices.

`iscale`
: Numeric and positive. Optional initial value for σ. Recycled to the appropriate length. In general, a larger value is better than a smaller value. A `NULL` means an initial value is computed internally.

`R`
: Numeric. Maximum number of values possible. See **Details** for more details.

`percentiles`
: Numeric vector of percentiles used for the fitted values. Values should be between 0 and 100. This argument uses the argument `R` if assigned. If `percentiles = NULL` then the mean will be returned as the fitted values.

`mpv`
: Logical. If `mpv = TRUE` then the *median predicted value* (MPV) is computed and returned as the (last) column of the fitted values. This argument is ignored if `percentiles = NULL`. See **Details** for more details.

`zero`
: A vector specifying which linear/additive predictors are modelled as intercepts only. The value (possibly values) can be from the set {1, 2} corresponding respectively to μ and σ. By default all linear/additive predictors are modelled as a linear combination of the explanatory variables. See `CommonVGAMffArguments` for more details.

Details

The Gumbel distribution is a generalized extreme value (GEV) distribution with *shape* parameter $\xi = 0$. Consequently it is more easily estimated than the GEV. See gev for more details.

The quantity R is the maximum number of observations possible, for example, in the Venice data below, the top 10 daily values are recorded for each year, therefore $R = 365$ because there are about 365 days per year. The MPV is the value of the response such that the probability of obtaining a value greater than the MPV is 0.5 out of R observations. For the Venice data, the MPV is the sea level such that there is an even chance that the highest level for a particular year exceeds the MPV. If `mpv = TRUE` then the column labelled `"MPV"` contains the MPVs when `fitted()` is applied to the fitted object.

The formula for the mean of a response Y is $\mu + \sigma \times Euler$ where $Euler$ is a constant that has value approximately equal to 0.5772. The formula for the percentiles are (if `R` is not given) $\mu - \sigma \times \log[-\log(P/100)]$ where P is the `percentile` argument value(s). If `R` is given then the percentiles are $\mu - \sigma \times \log[R(1 - P/100)]$.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Warning

When `R` is not given (the default) the fitted percentiles are that of the data, and not of the overall population. For example, in the example below, the 50 percentile is approximately the running median through the data, however, the data are the highest sea level measurements recorded each year (it therefore equates to the median predicted value or MPV).

Note

Like many other usual **VGAM** family functions, `gumbelff()` handles (independent) multiple responses.

`gumbel()` can handle more of a multivariate response, i.e., a matrix with more than one column. Each row of the matrix is sorted into descending order. Missing values in the response are allowed but require `na.action = na.pass`. The response matrix needs to be padded with any missing values. With a multivariate response one has a matrix `y`, say, where `y[, 2]` contains the second order statistics, etc.

Author(s)

T. W. Yee

References

Yee, T. W. and Stephenson, A. G. (2007). Vector generalized linear and additive extreme value models. *Extremes*, **10**, 1–19.

Smith, R. L. (1986). Extreme value theory based on the *r* largest annual events. *Journal of Hydrology*, **86**, 27–43.

Rosen, O. and Cohen, A. (1996). Extreme percentile regression. In: Haerdle, W. and Schimek, M. G. (eds.), *Statistical Theory and Computational Aspects of Smoothing: Proceedings of the COMPSTAT '94 Satellite Meeting held in Semmering, Austria, 27–28 August 1994*, pp.200–214, Heidelberg: Physica-Verlag.

Coles, S. (2001). *An Introduction to Statistical Modeling of Extreme Values*. London: Springer-Verlag.

See Also

rgumbel, dgumbelII, cens.gumbel, guplot, gev, gevff, venice.

Examples

```
# Example 1: Simulated data
gdata <- data.frame(y1 = rgumbel(n = 1000, loc = 100, scale = exp(1)))
fit1 <- vglm(y1 ~ 1, gumbelff(perc = NULL), data = gdata, trace = TRUE)
coef(fit1, matrix = TRUE)
Coef(fit1)
head(fitted(fit1))
with(gdata, mean(y1))

# Example 2: Venice data
(fit2 <- vglm(cbind(r1, r2, r3, r4, r5) ~ year, data = venice,
              gumbel(R = 365, mpv = TRUE), trace = TRUE))
head(fitted(fit2))
coef(fit2, matrix = TRUE)
sqrt(diag(vcov(summary(fit2))))   # Standard errors

# Example 3: Try a nonparametric fit ---------------------
# Use the entire data set, including missing values
# Same as as.matrix(venice[, paste0("r", 1:10)]):
Y <- Select(venice, "r", sort = FALSE)
fit3 <- vgam(Y ~ s(year, df = 3), gumbel(R = 365, mpv = TRUE),
             data = venice, trace = TRUE, na.action = na.pass)
depvar(fit3)[4:5, ]  # NAs used to pad the matrix

## Not run:   # Plot the component functions
par(mfrow = c(2, 3), mar = c(6, 4, 1, 2) + 0.3, xpd = TRUE)
plot(fit3, se = TRUE, lcol = "blue", scol = "limegreen", lty = 1,
     lwd = 2, slwd = 2, slty = "dashed")

# Quantile plot --- plots all the fitted values
qtplot(fit3, mpv = TRUE, lcol = c(1, 2, 5), tcol = c(1, 2, 5), lwd = 2,
       pcol = "blue", tadj = 0.1, ylab = "Sea level (cm)")

# Plot the 99 percentile only
year <- venice[["year"]]
matplot(year, Y, ylab = "Sea level (cm)", type = "n")
matpoints(year, Y, pch = "*", col = "blue")
lines(year, fitted(fit3)[, "99%"], lwd = 2, col = "orange")

# Check the 99 percentiles with a smoothing spline.
```

```
# Nb. (1-0.99) * 365 = 3.65 is approx. 4, meaning the 4th order
# statistic is approximately the 99 percentile.
plot(year, Y[, 4], ylab = "Sea level (cm)", type = "n",
     main = "Orange is 99 percentile, Green is a smoothing spline")
points(year, Y[, 4], pch = "4", col = "blue")
lines(year, fitted(fit3)[, "99%"], lty = 1, col = "orange")
lines(smooth.spline(year, Y[, 4], df = 4), col = "limegreen", lty = 2)

## End(Not run)
```

Gumbel-II *The Gumbel-II Distribution*

Description

Density, cumulative distribution function, quantile function and random generation for the Gumbel-II distribution.

Usage

```
dgumbelII(x, scale = 1, shape, log = FALSE)
pgumbelII(q, scale = 1, shape, lower.tail = TRUE, log.p = FALSE)
qgumbelII(p, scale = 1, shape, lower.tail = TRUE, log.p = FALSE)
rgumbelII(n, scale = 1, shape)
```

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
n	number of observations. Same as in runif.
log	Logical. If log = TRUE then the logarithm of the density is returned.
lower.tail, log.p	Same meaning as in pnorm or qnorm.
shape, scale	positive shape and scale parameters.

Details

See gumbelII for details.

Value

dgumbelII gives the density, pgumbelII gives the cumulative distribution function, qgumbelII gives the quantile function, and rgumbelII generates random deviates.

Author(s)

T. W. Yee and Kai Huang

See Also

gumbelII, dgumbel.

Examples

```
probs <- seq(0.01, 0.99, by = 0.01)
Scale <- exp(1); Shape <- exp( 0.5);
max(abs(pgumbelII(qgumbelII(p = probs, shape = Shape, Scale),
                  shape = Shape, Scale) - probs))  # Should be 0

## Not run:  x <- seq(-0.1, 10, by = 0.01);
plot(x, dgumbelII(x, shape = Shape, Scale), type = "l", col = "blue",
     main = "Blue is density, orange is the CDF", las = 1,
     sub = "Red lines are the 10,20,...,90 percentiles",
     ylab = "", ylim = 0:1)
abline(h = 0, col = "blue", lty = 2)
lines(x, pgumbelII(x, shape = Shape, Scale), col = "orange")
probs <- seq(0.1, 0.9, by = 0.1)
Q <- qgumbelII(probs, shape = Shape, Scale)
lines(Q, dgumbelII(Q, Scale, Shape), col = "red", lty = 3, type = "h")
pgumbelII(Q, shape = Shape, Scale) - probs # Should be all zero
abline(h = probs, col = "red", lty = 3)
## End(Not run)
```

---

gumbelII *Gumbel-II Regression Family Function*

Description

Maximum likelihood estimation of the 2-parameter Gumbel-II distribution.

Usage

```
gumbelII(lscale = "loglink", lshape = "loglink", iscale = NULL, ishape = NULL,
         probs.y = c(0.2, 0.5, 0.8), perc.out = NULL, imethod = 1,
         zero = "shape", nowarning = FALSE)
```

Arguments

nowarning	Logical. Suppress a warning?
lshape, lscale	Parameter link functions applied to the (positive) shape parameter (called s below) and (positive) scale parameter (called b below). See Links for more choices.

Parameter link functions applied to the

ishape, iscale	Optional initial values for the shape and scale parameters.
imethod	See weibullR.

`zero, probs.y` Details at `CommonVGAMffArguments`.

`perc.out` If the fitted values are to be quantiles then set this argument to be the percentiles of these, e.g., 50 for median.

Details

The Gumbel-II density for a response Y is

$$f(y; b, s) = sy^{s-1} \exp[-(y/b)^s]/(b^s)$$

for $b > 0$, $s > 0$, $y > 0$. The cumulative distribution function is

$$F(y; b, s) = \exp[-(y/b)^{-s}].$$

The mean of Y is $b\,\Gamma(1 - 1/s)$ (returned as the fitted values) when $s > 1$, and the variance is $b^2\,\Gamma(1 - 2/s)$ when $s > 2$. This distribution looks similar to `weibullR`, and is due to Gumbel (1954).

This **VGAM** family function currently does not handle censored data. Fisher scoring is used to estimate the two parameters. Probably similar regularity conditions hold for this distribution compared to the Weibull distribution.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Note

See `weibullR`. This **VGAM** family function handles multiple responses.

Author(s)

T. W. Yee

References

Gumbel, E. J. (1954). Statistical theory of extreme values and some practical applications. *Applied Mathematics Series*, volume 33, U.S. Department of Commerce, National Bureau of Standards, USA.

See Also

`dgumbelII`, `gumbel`, `gev`.

Examples

```
gdata <- data.frame(x2 = runif(nn <- 1000))
gdata <- transform(gdata, heta1  = +1,
                          heta2  = -1 + 0.1 * x2,
                          ceta1 =  0,
                          ceta2 =  1)
```

```
gdata <- transform(gdata, shape1 = exp(heta1),
                          shape2 = exp(heta2),
                          scale1 = exp(ceta1),
                          scale2 = exp(ceta2))
gdata <- transform(gdata,
                   y1 = rgumbelII(nn, scale = scale1, shape = shape1),
                   y2 = rgumbelII(nn, scale = scale2, shape = shape2))

fit <- vglm(cbind(y1, y2) ~ x2,
            gumbelII(zero = c(1, 2, 3)), data = gdata, trace = TRUE)
coef(fit, matrix = TRUE)
vcov(fit)
summary(fit)
```

---

gumbelUC *The Gumbel Distribution*

Description

Density, distribution function, quantile function and random generation for the Gumbel distribution with location parameter `location` and scale parameter `scale`.

Usage

```
dgumbel(x, location = 0, scale = 1, log = FALSE)
pgumbel(q, location = 0, scale = 1, lower.tail = TRUE, log.p = FALSE)
qgumbel(p, location = 0, scale = 1, lower.tail = TRUE, log.p = FALSE)
rgumbel(n, location = 0, scale = 1)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. If `length(n)` > 1 then the length is taken to be the number required.
`location`	the location parameter μ. This is not the mean of the Gumbel distribution (see **Details** below).
`scale`	the scale parameter σ. This is not the standard deviation of the Gumbel distribution (see **Details** below).
`log`	Logical. If `log = TRUE` then the logarithm of the density is returned.
`lower.tail, log.p`	Same meaning as in `punif` or `qunif`.

Details

The Gumbel distribution is a special case of the *generalized extreme value* (GEV) distribution where the shape parameter ξ = 0. The latter has 3 parameters, so the Gumbel distribution has two. The Gumbel distribution function is

$$G(y) = \exp\left(-\exp\left[-\frac{y-\mu}{\sigma}\right]\right)$$

where $-\infty < y < \infty$, $-\infty < \mu < \infty$ and $\sigma > 0$. Its mean is

$$\mu - \sigma * \gamma$$

and its variance is

$$\sigma^2 * \pi^2/6$$

where γ is Euler's constant (which can be obtained as `-digamma(1)`).

See `gumbel`, the **VGAM** family function for estimating the two parameters by maximum likelihood estimation, for formulae and other details. Apart from `n`, all the above arguments may be vectors and are recyled to the appropriate length if necessary.

Value

`dgumbel` gives the density, `pgumbel` gives the distribution function, `qgumbel` gives the quantile function, and `rgumbel` generates random deviates.

Note

The **VGAM** family function `gumbel` can estimate the parameters of a Gumbel distribution using maximum likelihood estimation.

Author(s)

T. W. Yee

References

Coles, S. (2001). *An Introduction to Statistical Modeling of Extreme Values*. London: Springer-Verlag.

See Also

`gumbel`, `gumbelff`, `gev`, `dgompertz`.

Examples

```
mu <- 1; sigma <- 2;
y <- rgumbel(n = 100, loc = mu, scale = sigma)
c(mean(y), mu - sigma * digamma(1))  # Sample and population means
c(var(y), sigma^2 * pi^2 / 6)  # Sample and population variances

## Not run:  x <- seq(-2.5, 3.5, by = 0.01)
```

```
loc <- 0; sigma <- 1
plot(x, dgumbel(x, loc, sigma), type = "l", col = "blue",
     main = "Blue is density, red is the CDF", ylim = c(0, 1),
     sub = "Purple are 5,10,...,95 percentiles", ylab = "", las = 1)
abline(h = 0, col = "blue", lty = 2)
lines(qgumbel(seq(0.05, 0.95, by = 0.05), loc, sigma),
  dgumbel(qgumbel(seq(0.05, 0.95, by = 0.05), loc, sigma), loc, sigma),
      col = "purple", lty = 3, type = "h")
lines(x, pgumbel(x, loc, sigma), type = "l", col = "red")
abline(h = 0, lty = 2)
## End(Not run)
```

---

guplot *Gumbel Plot*

Description

Produces a Gumbel plot, a diagnostic plot for checking whether the data appears to be from a Gumbel distribution.

Usage

```
guplot(object, ...)
guplot.default(y, main = "Gumbel Plot",
    xlab = "Reduced data", ylab = "Observed data", type = "p", ...)
guplot.vlm(object, ...)
```

Arguments

y	A numerical vector. NAs etc. are not allowed.
main	Character. Overall title for the plot.
xlab	Character. Title for the x axis.
ylab	Character. Title for the y axis.
type	Type of plot. The default means points are plotted.
object	An object that inherits class "vlm", usually of class vglm-class or vgam-class.
...	Graphical argument passed into plot. See par for an exhaustive list. The arguments xlim and ylim are particularly useful.

Details

If Y has a Gumbel distribution then plotting the sorted values y_i versus the *reduced values* r_i should appear linear. The reduced values are given by

$$r_i = -\log(-\log(p_i))$$

where p_i is the ith plotting position, taken here to be $(i - 0.5)/n$. Here, n is the number of observations. Curvature upwards/downwards may indicate a Frechet/Weibull distribution, respectively. Outliers may also be detected using this plot.

The function guplot is generic, and guplot.default and guplot.vlm are some methods functions for Gumbel plots.

Value

A list is returned invisibly with the following components.

x	The reduced data.

y	The sorted y data.

Note

The Gumbel distribution is a special case of the GEV distribution with shape parameter equal to zero.

Author(s)

T. W. Yee

References

Coles, S. (2001). *An Introduction to Statistical Modeling of Extreme Values*. London: Springer-Verlag.

Gumbel, E. J. (1958). *Statistics of Extremes*. New York, USA: Columbia University Press.

See Also

gumbel, gumbelff, gev, venice.

Examples

```
## Not run: guplot(rnorm(500), las = 1) -> ii
names(ii)

guplot(with(venice, r1), col = "blue")  # Venice sea levels data

## End(Not run)
```

`has.interceptvlm` *Has a Fitted VGLM Got an Intercept Term?*

Description

Looks at the `formula` to see if it has an intercept term.

Usage

```
has.intercept(object, ...)
has.interceptvlm(object, form.number = 1, ...)
```

Arguments

`object`	A fitted model object.
`form.number`	Formula number, is 1 or 2. which correspond to the arguments `formula` and `form2` respectively.
`...`	Arguments that are might be passed from one function to another.

Details

This methods function is a simple way to determine whether a fitted `vglm` object etc. has an intercept term or not. It is not entirely foolproof because one might suppress the intercept from the formula and then add in a variable in the formula that has a constant value.

Value

Returns a single logical.

Author(s)

Thomas W. Yee

See Also

`formulavlm`, `termsvlm`.

Examples

```
# Example: this is based on a glm example
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3, 1, 9); treatment <- gl(3, 3)
pdata <- data.frame(counts, outcome, treatment)  # Better style
vglm.D93 <- vglm(counts ~ outcome + treatment, poissonff, data = pdata)
formula(vglm.D93)
term.names(vglm.D93)
responseName(vglm.D93)
has.intercept(vglm.D93)
```

---

hatvalues — *Hat Values and Regression Deletion Diagnostics*

Description

When complete, a suite of functions that can be used to compute some of the regression (leave-one-out deletion) diagnostics, for the VGLM class.

Usage

```
hatvalues(model, ...)
hatvaluesvlm(model, type = c("diagonal", "matrix", "centralBlocks"), ...)
hatplot(model, ...)
hatplot.vlm(model, multiplier = c(2, 3), lty = "dashed",
            xlab = "Observation", ylab = "Hat values", ylim = NULL, ...)
dfbetavlm(model, maxit.new = 1,
          trace.new = FALSE,
          smallno = 1.0e-8, ...)
```

Arguments

`model` an R object, typically returned by `vglm`.

`type` Character. The default is the first choice, which is a $nM \times nM$ matrix. If `type = "matrix"` then the *entire* hat matrix is returned. If `type = "centralBlocks"` then n central $M \times M$ block matrices, in matrix-band format.

`multiplier` Numeric, the multiplier. The usual rule-of-thumb is that values greater than two or three times the average leverage (at least for the linear model) should be checked.

`lty, xlab, ylab, ylim`
Graphical parameters, see `par` etc. The default of `ylim` is `c(0, max(hatvalues(model)))` which means that if the horizontal dashed lines cannot be seen then there are no particularly influential observations.

`maxit.new, trace.new, smallno`
Having `maxit.new = 1` will give a one IRLS step approximation from the ordinary solution (and no warnings!). Else having `maxit.new = 10`, say, should usually mean convergence will occur for all observations when they are removed one-at-a-time. Else having `maxit.new = 2`, say, should usually mean some lack of convergence will occur when observations are removed one-at-a-time. Setting `trace.new = TRUE` will produce some running output at each IRLS iteration and for each individual row of the model matrix. The argument `smallno` multiplies each value of the original prior weight (often unity); setting it identically to zero will result in an error, but setting a very small value effectively removes that observation.

`...` further arguments, for example, graphical parameters for `hatplot.vlm()`.

Details

The invocation `hatvalues(vglmObject)` should return a $n \times M$ matrix of the diagonal elements of the hat (projection) matrix of a `vglm` object. To do this, the QR decomposition of the object is retrieved or reconstructed, and then straightforward calculations are performed.

The invocation `hatplot(vglmObject)` should plot the diagonal of the hat matrix for each of the M linear/additive predictors. By default, two horizontal dashed lines are added; hat values higher than these ought to be checked.

Note

It is hoped, soon, that the full suite of functions described at influence.measures will be written for VGLMs. This will enable general regression deletion diagnostics to be available for the entire VGLM class.

Author(s)

T. W. Yee.

See Also

vglm, cumulative, influence.measures.

Examples

```
# Proportional odds model, p.179, in McCullagh and Nelder (1989)
pneumo <- transform(pneumo, let = log(exposure.time))
fit <- vglm(cbind(normal, mild, severe) ~ let, cumulative, data = pneumo)
hatvalues(fit)  # n x M matrix, with positive values
all.equal(sum(hatvalues(fit)), fit@rank)  # Should be TRUE
## Not run:  par(mfrow = c(1, 2))
hatplot(fit, ylim = c(0, 1), las = 1, col = "blue")
## End(Not run)
```

---

hdeff *Hauck-Donner Effects: A Detection Test for Wald Tests*

Description

A detection test for the Hauck-Donner effect on each regression coefficient of a VGLM regression or 2 x 2 table.

Usage

```
hdeff(object, ...)
hdeff.vglm(object, derivative = NULL, se.arg = FALSE,
           subset = NULL, theta0 = 0, hstep = 0.005,
           fd.only = FALSE, ...)
hdeff.numeric(object, byrow = FALSE, ...)
hdeff.matrix(object, ...)
```

Arguments

object Usually a vglm object. Although only a limited number of family functions have an analytical solution to the HDE detection test (binomialff, borel.tanner, cumulative, erlang, felix, lindley, poissonff, topple, uninormal, zipoissonff, and zipoisson; hopefully some more will be implemented in the short future!)

the finite-differences (FDs) method can be applied to almost all **VGAM** family functions to get a numerical solution.

Alternatively `object` may represent a 2 x 2 table of *positive* counts. If so, then the first row corresponds to $x2 = 0$ (baseline group) and the second row $x2 = 1$. The first column corresponds to $y = 0$ (failure) and the second column $y = 1$ (success).

Another alternative is that `object` is a numerical vector of length 4, representing a 2 x 2 table of *positive* counts. If so then it is fed into `hdeff.matrix` using the argument `byrow`, which matches `matrix`. See the examples below.

`derivative` Numeric. Either 1 or 2. Currently only a few models having one linear predictor are handled analytically for `derivative = 2`, e.g., `binomialff`, `poissonff`. However, the numerical method can return the first two derivatives for almost all models.

`se.arg` Logical. If `TRUE` then the derivatives of the standard errors are returned as well, because usually the derivatives of the Wald statistics are of central interest. Requires `derivative` to be assigned the value 1 or 2 for this argument to operate.

`subset` Logical or vector of indices, to select the regression coefficients of interest. The default is to select all coefficients. Recycled if necessary if logical. If numeric then they should comprise elements from `1:length(coef(object))`. This argument can be useful for computing the derivatives of a Cox regression (`coxph`) fitted using artificially created Poisson data; then there are many coefficients that are effectively nuisance parameters.

`theta0` Numeric. Vector recycled to the necessary length which is the number of regression coefficients. The null hypotheses for the regression coefficients are that they equal those respective values, and the alternative hypotheses are all two-sided. It is not recommended that argument `subset` be used if a vector of values is assigned here because `theta0[subset]` is implied and might not work.

`hstep` Positive numeric and recycled to length 2; it is the so-called *step size* when using finite-differences and is often called h in the calculus literature, e.g., $f'(x)$ is approximately $(f(x+h) - f(x))/h$. For the 2nd-order partial derivatives, there are two step sizes and hence this argument is recycled to length 2. The default is to have the same values. The 1st-order derivatives use the first value only. It is recommended that a few values of this argument be tried because values of the first and second derivatives can vary accordingly. If any values are too large then the derivatives may be inaccurate; and if too small then the derivatives may be unstable and subject to too much round-off/cancellation error (in fact it may create an error or a `NA`).

`fd.only` Logical; if `TRUE` then finite-differences are used to estimate the derivatives even if an analytical solution has been coded, By default, finite-differences will be used when an analytical solution has not been implemented.

It is possible that `NA`s are returned. If so, and if `fd.only = FALSE`, then a warning is issued and a recursive call is made with `fd.only = TRUE`—this is more likely to return an answer without any `NA`s.

`byrow` Logical; fed into `matrix` if `object` is a vector of length 4 so that there are two choices in the order of the elements.

`...` currently unused but may be used in the future for further arguments passed into the other methods functions.

Details

Almost all of statistical inference based on the likelihood assumes that the parameter estimates are located in the interior of the parameter space. The nonregular case of being located on the boundary is not considered very much and leads to very different results from the regular case. Practically, an important question is: how close is close to the boundary? One might answer this as: the parameter estimates are too close to the boundary when the Hauck-Donner effect (HDE) is present, whereby the Wald statistic becomes aberrant.

Hauck and Donner (1977) first observed an aberration of the Wald test statistic not monotonically increasing as a function of increasing distance between the parameter estimate and the null value. This "disturbing" and "undesirable" underappreciated effect has since been observed in other regression models by various authors. This function computes the first, and possibly second, derivative of the Wald statistic for each regression coefficient. A negative value of the first derivative is indicative of the HDE being present. More information can be obtained from `hdeffsev` regarding HDE severity: there may be none, faint, weak, moderate, strong and extreme amounts of HDE present.

In general, most models have derivatives that are computed numerically using finite-difference approximations. The reason is that it takes a lot of work to program in the analytical solution (this includes a few very common models, such as `poissonff` and `binomialff`, where the first two derivatives have been implemented).

Value

By default this function returns a labelled logical vector; a `TRUE` means the HDE is affirmative for that coefficient (negative slope). Hence ideally all values are `FALSE`. Any `TRUE` values suggests that the MLE is too near the boundary of the parameter space, and that the p-value for that regression coefficient is biased upwards. When present a highly significant variable might be deemed nonsignificant, and thus the HDE can create havoc for variable selection. If the HDE is present then more accurate p-values can generally be obtained by conducting a likelihood ratio test (see `lrt.stat.vlm`) or Rao's score test (see `score.stat.vlm`); indeed the default of `wald.stat.vlm` does not suffer from the HDE.

Setting `deriv = 1` returns a numerical vector of first derivatives of the Wald statistics. Setting `deriv = 2` returns a 2-column matrix of first and second derivatives of the Wald statistics. Then setting `se.arg = TRUE` returns an additional 1 or 2 columns.

Some 2nd derivatives are `NA` if only a partial analytic solution has been programmed in.

For those **VGAM** family functions whose HDE test has not yet been implemented explicitly (the vast majority of them), finite-difference approximations to the derivatives will be used—see the arguments `hstep` and `fd.only` for getting some control on them.

Note

The function `summaryvglm` conducts the HDE detection test if possible and prints out a line at the bottom if the HDE is detected for some regression coefficients. By "if possible", only a few family functions are exempt and they have an `infos` slot with component `hadof = FALSE`; such as `normal.vcm`, `rec.normal` because it uses the BFGS-IRLS method for computing the working weights. For these few a `NULL` is returned by `hdeff`.

If the second derivatives are of interest then it is recommended that `crit = "c"` be added to the fitting so that a slightly more accurate model results (usually one more IRLS iteration). This is because the FD approximation is very sensitive to values of the working weights, so they need to be computed accurately. Occasionally, if the coefficient is close to 0, then its Wald statistic's second derivative may be unusually large in magnitude (this could be due to something such as roundoff error).

This function is currently under development and may change a little in the short future. For HDE severity measures see `hdeffsev`.

Author(s)

Thomas W. Yee.

References

Hauck, J. W. W. and A. Donner (1977). Wald's test as applied to hypotheses in logit analysis. *Journal of the American Statistical Association*, **72**, 851–853.

Yee, T. W. (2022). On the Hauck-Donner effect in Wald tests: Detection, tipping points and parameter space characterization, *Journal of the American Statistical Association*, **117**, 1763–1774. doi:10.1080/01621459.2021.1886936.

Yee, T. W. (2021). Some new results concerning the Hauck-Donner effect. *Manuscript in preparation*.

See Also

`summaryvglm`, `hdeffsev`, `alogitlink`, `asinlink`, `vglm`, `lrt.stat`, `score.stat`, `wald.stat`, `confintvglm`, `profilevglm`.

Examples

```
pneumo <- transform(pneumo, let = log(exposure.time))
fit <- vglm(cbind(normal, mild, severe) ~ let, data = pneumo,
            trace = TRUE, crit = "c",  # Get some more accuracy
            cumulative(reverse = TRUE,  parallel = TRUE))
cumulative()@infos()$hadof  # Analytical solution implemented
hdeff(fit)
hdeff(fit, deriv = 1)  # Analytical solution
hdeff(fit, deriv = 2)  # It is a partial analytical solution
hdeff(fit, deriv = 2, se.arg = TRUE,
      fd.only = TRUE)  # All derivatives solved numerically by FDs

# 2 x 2 table of counts
R0 <- 25; N0 <- 100  # Hauck Donner (1977) data set
mymat <- c(N0-R0, R0, 8, 92)  # HDE present
(mymat <- matrix(mymat, 2, 2, byrow = TRUE))
hdeff(mymat)
hdeff(c(mymat))  # Input is a vector
hdeff(c(t(mymat)), byrow = TRUE)  # Reordering of the data
```

hdeffsev *Hauck-Donner Effects: Severity Measures*

Description

Computes the severity of the Hauck-Donner effect for each regression coefficient of a VGLM regression.

Usage

```
hdeffsev(x, y, dy, ddy, allofit = FALSE, eta0 = 0, COPS0 = eta0,
         severity.table = c("None", "Faint", "Weak",
             "Moderate", "Strong", "Extreme", "Undetermined"))
```

Arguments

`x, y`	Numeric vectors; `x` are the estimates (sorted), and `y` are the signed Wald statistics.
`dy, ddy`	Numeric vectors; the first and second derivatives of the Wald statistics. They can be computed by `hdeff`.
`allofit`	Logical. If `TRUE` then other quantities are returned in a list. The default is a vector with elements selected from the argument `severity.table`.
`severity.table`	Character vector with 6 values. The last value is used for initialization. Usually users should not assign anything to this argument.
`eta0`	Numeric. The hypothesized value. The default is appropriate for most symmetric binomial links,and also for Poisson regression with the natural parameter.
`COPS0`	Numeric. See Yee (2021).

Details

This function is rough-and-ready. It is possible to use the first two derivatives obtained from `hdeff` to categorize the severity of the the Hauck-Donner effect (HDE). It is effectively assumed that, starting at the origin and going right, the curve is made up of a convex segment followed by a concave segment and then the convex segment. Midway in the concave segment the derivative is 0, and beyond that the HDE is really manifest because the derivative is negative.

For `"none"` the estimate lies on the convex part of the curve near the origin, hence there is very little HDE at all.

For `"weak"` the estimate lies on the concave part of the curve but the Wald statistic is still increasing as estimate gets away from 0, hence it is only a mild form of the HDE.

Previously `"faint"` was used but now it has been omitted.

For `"moderate"`, `"strong"` and `"extreme"` the Wald statistic is decreasing as the estimate gets away from `eta0`, hence it really does exhibit the HDE. It is recommended that `lrt.stat` be used to compute LRT p-values, as they do not suffer from the HDE.

Value

By default this function returns a labelled vector with elements selected from `severity.table`. If `allofit = TRUE` then Yee (2022) gives details about some of the other list components, e.g., a quantity called `zeta` is the normal line projected onto the x-axis, and its first derivative gives additional information about the position of the estimate along the curve.

Note

This function is likely to change in the short future because it is experimental and far from complete. Improvements are intended.

Currently, in order for "Strong" to be assigned correctly, at least one such value is needed on the LHS and/or RHS each. From those, two other boundary points are obtained so that it creates two intervals.

Author(s)

Thomas W. Yee.

References

Yee, T. W. (2022). On the Hauck-Donner effect in Wald tests: Detection, tipping points and parameter space characterization, *Journal of the American Statistical Association*, **117**, 1763–1774. doi:10.1080/01621459.2021.1886936.

Yee, T. W. (2022). Some new results concerning the Wald tests and the parameter space. *In review*.

See Also

`seglines`, `hdeff`.

Examples

```
deg <- 4  # myfun is a function that approximates the HDE
myfun <- function(x, deriv = 0) switch(as.character(deriv),
  '0' = x^deg * exp(-x),
  '1' = (deg * x^(deg-1) - x^deg) * exp(-x),
  '2' = (deg*(deg-1)*x^(deg-2) - 2*deg*x^(deg-1) + x^deg)*exp(-x))

xgrid <- seq(0, 10, length = 101)
ansm <- hdeffsev(xgrid, myfun(xgrid), myfun(xgrid, deriv = 1),
                 myfun(xgrid, deriv = 2), allofit = TRUE)
digg <- 4
cbind(severity = ansm$sev,
      fun      = round(myfun(xgrid), digg),
      deriv1   = round(myfun(xgrid, deriv = 1), digg),
      deriv2   = round(myfun(xgrid, deriv = 2), digg),
      zderiv1  = round(1 + (myfun(xgrid, deriv = 1))^2 +
                       myfun(xgrid, deriv = 2) * myfun(xgrid), digg))
```

hormone *Hormone Assay Data*

Description

A hormone assay data set from Carroll and Ruppert (1988).

Usage

```
data(hormone)
```

Format

A data frame with 85 observations on the following 2 variables.

`X` a numeric vector, suitable as the x-axis in a scatter plot. The reference method.

`Y` a numeric vector, suitable as the y-axis in a scatter plot. The test method.

Details

The data is given in Table 2.4 of Carroll and Ruppert (1988), and was downloaded from `http://www.stat.tamu.edu/~carr` prior to 2019. The book describes the data as follows. The data are the results of two assay methods for hormone data; the scale of the data as presented is not particularly meaningful, and the original source of the data refused permission to divulge further information. As in a similar example of Leurgans (1980), the old or reference method is being used to predict the new or test method. The overall goal is to see whether we can reproduce the test-method measurements with the reference-method measurements. Thus calibration might be of interest for the data.

References

Carroll, R. J. and Ruppert, D. (1988). *Transformation and Weighting in Regression*. New York, USA: Chapman & Hall.

Leurgans, S. (1980). Evaluating laboratory measurement techniques. *Biostatistics Casebook*. Eds.: Miller, R. G. Jr., and Efron, B. and Brown, B. W. Jr., and Moses, L. New York, USA: Wiley.

Yee, T. W. (2014). Reduced-rank vector generalized linear models with two linear predictors. *Computational Statistics and Data Analysis*, **71**, 889–902.

See Also

`uninormal`, `rrvglm`.

Examples

```
## Not run:
data(hormone)
summary(hormone)

modelI <-rrvglm(Y ~ 1 + X, data = hormone, trace = TRUE,
                uninormal(zero = NULL, lsd = "identitylink", imethod = 2))

# Alternative way to fit modelI
modelI.other <- vglm(Y ~ 1 + X, data = hormone, trace = TRUE,
                     uninormal(zero = NULL, lsd = "identitylink"))

# Inferior to modelI
modelII <- vglm(Y ~ 1 + X, data = hormone, trace = TRUE,
                family = uninormal(zero = NULL))

logLik(modelI)
logLik(modelII)  # Less than logLik(modelI)


# Reproduce the top 3 equations on p.65 of Carroll and Ruppert (1988).
# They are called Equations (1)--(3) here.

# Equation (1)
hormone <- transform(hormone, rX = 1 / X)
clist <- list("(Intercept)" = diag(2), X = diag(2), rX = rbind(0, 1))
fit1 <- vglm(Y ~ 1 + X + rX, family = uninormal(zero = NULL),
             constraints = clist, data = hormone, trace = TRUE)
coef(fit1, matrix = TRUE)
summary(fit1)  # Actually, the intercepts do not seem significant
plot(Y ~ X, hormone, col = "blue")
lines(fitted(fit1) ~ X, hormone, col = "orange")

# Equation (2)
fit2 <- rrvglm(Y ~ 1 + X, uninormal(zero = NULL), hormone, trace = TRUE)
coef(fit2, matrix = TRUE)
plot(Y ~ X, hormone, col = "blue")
lines(fitted(fit2) ~ X, hormone, col = "red")
# Add +- 2 SEs
lines(fitted(fit2) + 2 * exp(predict(fit2)[, "loglink(sd)"]) ~ X,
      hormone, col = "orange")
lines(fitted(fit2) - 2 * exp(predict(fit2)[, "loglink(sd)"]) ~ X,
      hormone, col = "orange")

# Equation (3)
# Does not fit well because the loglink link for the mean is not good.
fit3 <- rrvglm(Y ~ 1 + X, maxit = 300, data = hormone, trace = TRUE,
               uninormal(lmean = "loglink", zero = NULL))
coef(fit3, matrix = TRUE)
plot(Y ~ X, hormone, col = "blue")  # Does not look okay.
lines(exp(predict(fit3)[, 1]) ~ X, hormone, col = "red")
# Add +- 2 SEs
```

```
lines(fitted(fit3) + 2 * exp(predict(fit3)[, "loglink(sd)"]) ~ X,
      hormone, col = "orange")
lines(fitted(fit3) - 2 * exp(predict(fit3)[, "loglink(sd)"]) ~ X,
      hormone, col = "orange")

## End(Not run)
```

---

hspider *Hunting Spider Data*

Description

Abundance of hunting spiders in a Dutch dune area.

Usage

```
data(hspider)
```

Format

A data frame with 28 observations (sites) on the following 18 variables.

WaterCon Log percentage of soil dry mass.

BareSand Log percentage cover of bare sand.

FallTwig Log percentage cover of fallen leaves and twigs.

CoveMoss Log percentage cover of the moss layer.

CoveHerb Log percentage cover of the herb layer.

ReflLux Reflection of the soil surface with cloudless sky.

Alopacce Abundance of *Alopecosa accentuata*.

Alopcune Abundance of *Alopecosa cuneata*.

Alopfabr Abundance of *Alopecosa fabrilis*.

Arctlute Abundance of *Arctosa lutetiana*.

Arctperi Abundance of *Arctosa perita*.

Auloalbi Abundance of *Aulonia albimana*.

Pardlugu Abundance of *Pardosa lugubris*.

Pardmont Abundance of *Pardosa monticola*.

Pardnigr Abundance of *Pardosa nigriceps*.

Pardpull Abundance of *Pardosa pullata*.

Trocterr Abundance of *Trochosa terricola*.

Zoraspin Abundance of *Zora spinimana*.

Details

The data, which originally came from Van der Aart and Smeek-Enserink (1975) consists of abundances (numbers trapped over a 60 week period) and 6 environmental variables. There were 28 sites.

This data set has been often used to illustrate ordination, e.g., using canonical correspondence analysis (CCA). In the example below, the data is used for constrained quadratic ordination (CQO; formerly called canonical Gaussian ordination or CGO), a numerically intensive method that has many superior qualities. See cqo for details.

References

Van der Aart, P. J. M. and Smeek-Enserink, N. (1975). Correlations between distributions of hunting spiders (Lycosidae, Ctenidae) and environmental characteristics in a dune area. *Netherlands Journal of Zoology*, **25**, 1–45.

Examples

```
summary(hspider)

## Not run:
# Standardize the environmental variables:
hspider[, 1:6] <- scale(subset(hspider, select = WaterCon:ReflLux))

# Fit a rank-1 binomial CAO
hsbin <- hspider  # Binary species data
hsbin[, -(1:6)] <- as.numeric(hsbin[, -(1:6)] > 0)
set.seed(123)
ahsb1 <- cao(cbind(Alopcune, Arctlute, Auloalbi, Zoraspin) ~
             WaterCon + ReflLux,
             family = binomialff(multiple.responses = TRUE),
             df1.nl = 2.2, Bestof = 3, data = hsbin)
par(mfrow = 2:1, las = 1)
lvplot(ahsb1, type = "predictors", llwd = 2,
       ylab = "logitlink(p)", lcol = 1:9)
persp(ahsb1, rug = TRUE, col = 1:10, lwd = 2)
coef(ahsb1)

## End(Not run)
```

---

huber2 *Huber's Least Favourable Distribution Family Function*

Description

M-estimation of the two parameters of Huber's least favourable distribution. The one parameter case is also implemented.

Usage

```
huber1(llocation = "identitylink", k = 0.862, imethod = 1)
huber2(llocation = "identitylink", lscale = "loglink",
       k = 0.862, imethod = 1, zero = "scale")
```

Arguments

`llocation, lscale`
: Link functions applied to the location and scale parameters. See `Links` for more choices.

`k`
: Tuning constant. See `rhuber` for more information.

`imethod, zero`
: See `CommonVGAMffArguments` for information. The default value of `zero` means the scale parameter is modelled as intercept-only.

Details

Huber's least favourable distribution family function is popular for resistant/robust regression. The center of the distribution is normal and its tails are double exponential.

By default, the mean is the first linear/additive predictor (returned as the fitted values; this is the location parameter), and the log of the scale parameter is the second linear/additive predictor. The Fisher information matrix is diagonal; Fisher scoring is implemented.

The **VGAM** family function `huber1()` estimates only the location parameter. It assumes a scale parameter of unit value.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Note

Warning: actually, `huber2()` may be erroneous since the first derivative is not continuous when there are two parameters to estimate. `huber1()` is fine in this respect.

The response should be univariate.

Author(s)

T. W. Yee. Help was given by Arash Ardalan.

References

Huber, P. J. and Ronchetti, E. (2009). *Robust Statistics*, 2nd ed. New York: Wiley.

See Also

`rhuber`, `uninormal`, `laplace`, `CommonVGAMffArguments`.

Examples

```
set.seed(1231); NN <- 30; coef1 <- 1; coef2 <- 10
hdata <- data.frame(x2 = sort(runif(NN)))
hdata <- transform(hdata, y  = rhuber(NN, mu = coef1 + coef2 * x2))

hdata$x2[1] <- 0.0  # Add an outlier
hdata$y[1] <- 10

fit.huber2 <- vglm(y ~ x2, huber2(imethod = 3), hdata, trace = TRUE)
fit.huber1 <- vglm(y ~ x2, huber1(imethod = 3), hdata, trace = TRUE)

coef(fit.huber2, matrix = TRUE)
summary(fit.huber2)


## Not run:  # Plot the results
plot(y ~ x2, data = hdata, col = "blue", las = 1)
lines(fitted(fit.huber2) ~ x2, data = hdata, col = "darkgreen", lwd = 2)

fit.lm <- lm(y ~ x2, hdata)  # Compare to a LM:
lines(fitted(fit.lm) ~ x2, data = hdata, col = "lavender", lwd = 3)

# Compare to truth:
lines(coef1 + coef2 * x2 ~ x2, data = hdata, col = "orange",
      lwd = 2, lty = "dashed")

legend("bottomright", legend = c("truth", "huber", "lm"),
       col = c("orange", "darkgreen", "lavender"),
       lty = c("dashed", "solid", "solid"), lwd = c(2, 2, 3))
## End(Not run)
```

Huggins89.t1 *Table 1 of Huggins (1989)*

Description

Simulated capture data set for the linear logistic model depending on an occasion covariate and an individual covariate for 10 trapping occasions and 20 individuals.

Usage

```
data(Huggins89table1)
data(Huggins89.t1)
```

Format

The format is a data frame.

Details

Table 1 of Huggins (1989) gives this toy data set. Note that variables t1,...,t10 are occasion-specific variables. They correspond to the response variables y1,...,y10 which have values 1 for capture and 0 for not captured.

Both Huggins89table1 and Huggins89.t1 are identical. The latter used variables beginning with z, not t, and may be withdrawn very soon.

References

Huggins, R. M. (1989). On the statistical analysis of capture experiments. *Biometrika*, **76**, 133–140.

Examples

```
## Not run:
Huggins89table1 <-
  transform(Huggins89table1, x3.tij = t01,
            T02 = t02, T03 = t03, T04 = t04, T05 = t05, T06 = t06,
            T07 = t07, T08 = t08, T09 = t09, T10 = t10)
small.table1 <- subset(Huggins89table1,
  y01 + y02 + y03 + y04 + y05 + y06 + y07 + y08 + y09 + y10 > 0)
# fit.tbh is the bottom equation on p.133.
# It is a M_tbh model.
fit.tbh <-
  vglm(cbind(y01, y02, y03, y04, y05, y06, y07, y08, y09, y10) ~
       x2 + x3.tij,
       xij = list(x3.tij ~ t01 + t02 + t03 + t04 + t05 +
                           t06 + t07 + t08 + t09 + t10 +
                           T02 + T03 + T04 + T05 + T06 +
                           T07 + T08 + T09 + T10 - 1),
       posbernoulli.tb(parallel.t = TRUE ~ x2 + x3.tij),
       data = small.table1, trace = TRUE,
       form2 = ~  x2 + x3.tij +
                  t01 + t02 + t03 + t04 + t05 + t06 +
                  t07 + t08 + t09 + t10 +
                  T02 + T03 + T04 + T05 + T06 +
                  T07 + T08 + T09 + T10)

# These results differ a bit from Huggins (1989), probably because
# two animals had to be removed here (they were never caught):
coef(fit.tbh)  # First element is the behavioural effect
sqrt(diag(vcov(fit.tbh)))  # SEs
constraints(fit.tbh, matrix = TRUE)
summary(fit.tbh, presid = FALSE)
fit.tbh@extra$N.hat     # Estimate of the population site N; cf. 20.86
fit.tbh@extra$SE.N.hat  # Its standard error; cf. 1.87 or 4.51

fit.th <-
  vglm(cbind(y01, y02, y03, y04, y05, y06, y07, y08, y09, y10) ~ x2,
       posbernoulli.t, data = small.table1, trace = TRUE)
coef(fit.th)
constraints(fit.th)
```

```
coef(fit.th, matrix = TRUE)  # M_th model
summary(fit.th, presid = FALSE)
fit.th@extra$N.hat     # Estimate of the population size N
fit.th@extra$SE.N.hat  # Its standard error

fit.bh <-
  vglm(cbind(y01, y02, y03, y04, y05, y06, y07, y08, y09, y10) ~ x2,
       posbernoulli.b(I2 = FALSE), data = small.table1, trace = TRUE)
coef(fit.bh)
constraints(fit.bh)
coef(fit.bh, matrix = TRUE)  # M_bh model
summary(fit.bh, presid = FALSE)
fit.bh@extra$N.hat
fit.bh@extra$SE.N.hat

fit.h <-
  vglm(cbind(y01, y02, y03, y04, y05, y06, y07, y08, y09, y10) ~ x2,
       posbernoulli.b, data = small.table1, trace = TRUE)
coef(fit.h, matrix = TRUE)  # M_h model (version 1)
coef(fit.h)
summary(fit.h, presid = FALSE)
fit.h@extra$N.hat
fit.h@extra$SE.N.hat

Fit.h <-
  vglm(cbind(y01, y02, y03, y04, y05, y06, y07, y08, y09, y10) ~ x2,
       posbernoulli.t(parallel.t = TRUE ~ x2),
       data = small.table1, trace = TRUE)
coef(Fit.h)
coef(Fit.h, matrix = TRUE)  # M_h model (version 2)
summary(Fit.h, presid = FALSE)
Fit.h@extra$N.hat
Fit.h@extra$SE.N.hat

## End(Not run)
```

---

hunua *Hunua Ranges Data*

Description

The hunua data frame has 392 rows and 18 columns. Altitude is explanatory, and there are binary responses (presence/absence = 1/0 respectively) for 17 plant species.

Usage

```
data(hunua)
```

Format

This data frame contains the following columns:

agaaus Agathis australis, or Kauri

beitaw Beilschmiedia tawa, or Tawa

corlae Corynocarpus laevigatus

cyadea Cyathea dealbata

cyamed Cyathea medullaris

daccup Dacrydium cupressinum

dacdac Dacrycarpus dacrydioides

eladen Elaecarpus dentatus

hedarb Hedycarya arborea

hohpop Species name unknown

kniexc Knightia excelsa, or Rewarewa

kuneri Kunzea ericoides

lepsco Leptospermum scoparium

metrob Metrosideros robusta

neslan Nestegis lanceolata

rhosap Rhopalostylis sapida

vitluc Vitex lucens, or Puriri

altitude meters above sea level

Details

These were collected from the Hunua Ranges, a small forest in southern Auckland, New Zealand. At 392 sites in the forest, the presence/absence of 17 plant species was recorded, as well as the altitude. Each site was of area size $200m^2$.

Source

Dr Neil Mitchell, University of Auckland.

See Also

`waitakere`.

Examples

```
# Fit a GAM using vgam() and compare it with the Waitakere Ranges one
fit.h <- vgam(agaaus ~ s(altitude, df = 2), binomialff, data = hunua)
## Not run:
plot(fit.h, se = TRUE, lcol = "orange", scol = "orange",
     llwd = 2, slwd = 2, main = "Orange is Hunua, Blue is Waitakere")
## End(Not run)
```

```
head(predict(fit.h, hunua, type = "response"))

fit.w <- vgam(agaaus ~ s(altitude, df = 2), binomialff, data = waitakere)
## Not run:
plot(fit.w, se = TRUE, lcol = "blue", scol = "blue", add = TRUE)
## End(Not run)
head(predict(fit.w, hunua, type = "response"))   # Same as above?
```

Hurea *The Husler-Reiss Angular Surface Distribution*

Description

Density for the Husler-Reiss angular surface distribution.

Usage

```
dhurea(x, shape, log = FALSE)
```

Arguments

x	Same as `Uniform`.
shape	the positive (shape) parameter. It is often called $lambda$ and it might not be a shape parameter at all.
log	Logical. If `log = TRUE` then the logarithm of the density is returned.

Details

See `hurea`, the **VGAM** family function for estimating the (shape) parameter s by maximum likelihood estimation, for the formula of the probability density function.

Value

`dhurea` gives the density.

Warning

The cases `x == 0`, `x == 1`, `shape == 0` and `shape == Inf` may not be handled correctly.

Note

Difficulties are encountered as the shape parameter approaches 0 with respect to `integrate` because the density converges to a degenerate distribution with probability mass at 0 and 1. That is, when s is around 0.5 the density is "u" shaped and the area around the endpoints becomes concentrated at the two points. See the examples below. Approximately, the density is "u" shaped for $s < 1$ and unimodal for $s > 2$.

Author(s)

T. W. Yee

See Also

hurea.

Examples

```
integrate(dhurea, 0, 1, shape = 0.20)  # Incorrect
integrate(dhurea, 0, 1, shape = 0.35)  # struggling but okay
## Not run: x <- seq(0, 1, length = 501)
par(mfrow = c(2, 2))
plot(x, dhurea(x, 0.7), col = "blue", type = "l")
plot(x, dhurea(x, 1.1), col = "blue", type = "l")
plot(x, dhurea(x, 1.4), col = "blue", type = "l")
plot(x, dhurea(x, 3.0), col = "blue", type = "l")

## End(Not run)
```

---

hurea *Husler-Reiss Angular Surface Distribution Family Function*

Description

Estimating the parameter of the Husler-Reiss angular surface distribution by maximum likelihood estimation.

Usage

```
hurea(lshape = "loglink", zero = NULL, nrfs = 1,
      gshape = exp(3 * ppoints(5) - 1), parallel = FALSE)
```

Arguments

`lshape, gshape` Details at CommonVGAMffArguments.

`nrfs, zero, parallel`
Details at CommonVGAMffArguments.

Details

The Husler-Reiss angular surface distribution has a probability density function that can be written

$$f(y; s) = (s/(4 * sqrt(2 * pi) * y(1 - y)^2))exp(-(2 + s^2 * logity)^2/[8s^2])$$

for $0 < y < 1$ and positive shape parameter s. The mean of Y is currently unknown to me, as well as its quantiles. Hence s is currently returned as the fitted values. Fisher-scoring is implemented.

Value

An object of class "vglmff" (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Note

This **VGAM** family function handles multiple responses. It may struggle and/or fail when s is close to 0. Some comments about “u”-shaped versus unimodal densities accommodated by this distribution are at `dhurea`.

Author(s)

T. W. Yee

References

Mhalla, L. and de Carvalho, M. and Chavez-Demoulin, V. (2019). Regression-type models for extremal dependence. *Scandinavian Journal of Statistics*, **46**, 1141–1167.

See Also

`hurea`.

Examples

```
nn <- 100; set.seed(1)
hdata <- data.frame(x2 = runif(nn))
hdata <-
  transform(hdata,  # Cannot generate proper random variates!
    y1 = rbeta(nn, shape1 = 0.5, shape2 = 0.5),  # "U" shaped
    y2 = rnorm(nn, 0.65, sd = exp(-3 - 4 * x2)))
# Multiple responses:
hfit <- vglm(cbind(y1, y2) ~ x2, hurea, hdata, trace = TRUE)
coef(hfit, matrix = TRUE)
summary(hfit)
```

---

hyperg *Hypergeometric Family Function*

Description

Family function for a hypergeometric distribution where either the number of white balls or the total number of white and black balls are unknown.

Usage

```
hyperg(N = NULL, D = NULL, lprob = "logitlink", iprob = NULL)
```

Arguments

- `N` — Total number of white and black balls in the urn. Must be a vector with positive values, and is recycled, if necessary, to the same length as the response. One of `N` and `D` must be specified.
- `D` — Number of white balls in the urn. Must be a vector with positive values, and is recycled, if necessary, to the same length as the response. One of `N` and `D` must be specified.
- `lprob` — Link function for the probabilities. See `Links` for more choices.
- `iprob` — Optional initial value for the probabilities. The default is to choose initial values internally.

Details

Consider the scenario from `dhyper` where there are $N = m + n$ balls in an urn, where m are white and n are black. A simple random sample (i.e., *without* replacement) of k balls is taken. The response here is the sample *proportion* of white balls. In this document, `N` is $N = m + n$, `D` is m (for the number of "defectives", in quality control terminology, or equivalently, the number of marked individuals). The parameter to be estimated is the population proportion of white balls, viz. $prob = m/(m + n)$.

Depending on which one of `N` and `D` is inputted, the estimate of the other parameter can be obtained from the equation $prob = m/(m + n)$, or equivalently, `prob = D/N`. However, the log-factorials are computed using `lgamma` and both m and n are not restricted to being integer. Thus if an integer N is to be estimated, it will be necessary to evaluate the likelihood function at integer values about the estimate, i.e., at `trunc(Nhat)` and `ceiling(Nhat)` where `Nhat` is the (real) estimate of N.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, `vgam`, `rrvglm`, `cqo`, and `cao`.

Warning

No checking is done to ensure that certain values are within range, e.g., $k \leq N$.

Note

The response can be of one of three formats: a factor (first level taken as success), a vector of proportions of success, or a 2-column matrix (first column = successes) of counts. The argument `weights` in the modelling function can also be specified. In particular, for a general vector of proportions, you will need to specify `weights` because the number of trials is needed.

Author(s)

Thomas W. Yee

References

Forbes, C., Evans, M., Hastings, N. and Peacock, B. (2011). *Statistical Distributions*, Hoboken, NJ, USA: John Wiley and Sons, Fourth edition.

See Also

dhyper, binomialff.

Examples

```
nn <- 100
m <- 5  # Number of white balls in the population
k <- rep(4, len = nn)  # Sample sizes
n <- 4  # Number of black balls in the population
y  <- rhyper(nn = nn, m = m, n = n, k = k)
yprop <- y / k  # Sample proportions

# N is unknown, D is known. Both models are equivalent:
fit <- vglm(cbind(y,k-y) ~ 1, hyperg(D = m), trace = TRUE, crit = "c")
fit <- vglm(yprop ~ 1, hyperg(D = m), weight = k, trace = TRUE, crit = "c")

# N is known, D is unknown. Both models are equivalent:
fit <- vglm(cbind(y, k-y) ~ 1, hyperg(N = m+n), trace = TRUE, crit = "l")
fit <- vglm(yprop ~ 1, hyperg(N = m+n), weight = k, trace = TRUE, crit = "l")

coef(fit, matrix = TRUE)
Coef(fit)  # Should be equal to the true population proportion
unique(m / (m+n))  # The true population proportion
fit@extra
head(fitted(fit))
summary(fit)
```

---

hypersecant *Hyperbolic Secant Regression Family Function*

Description

Estimation of the parameter of the hyperbolic secant distribution.

Usage

```
  hypersecant(link.theta = extlogitlink(min = -pi/2, max = pi/2),
              init.theta = NULL)
hypersecant01(link.theta = extlogitlink(min = -pi/2, max = pi/2),
              init.theta = NULL)
```

Arguments

link.theta	Parameter link function applied to the parameter θ. See Links for more choices.
init.theta	Optional initial value for θ. If failure to converge occurs, try some other value. The default means an initial value is determined internally.

Details

The probability density function of the hyperbolic secant distribution is given by

$$f(y;\theta) = \exp(\theta y + \log(\cos(\theta)))/(2\cosh(\pi y/2)),$$

for parameter $-\pi/2 < \theta < \pi/2$ and all real y. The mean of Y is $\tan(\theta)$ (returned as the fitted values). Morris (1982) calls this model NEF-HS (Natural Exponential Family-Hyperbolic Secant). It is used to generate NEFs, giving rise to the class of NEF-GHS (G for Generalized).

Another parameterization is used for `hypersecant01()`: let $Y = (logit U)/\pi$. Then this uses

$$f(u;\theta) = (\cos(\theta)/\pi) \times u^{-0.5+\theta/\pi} \times (1-u)^{-0.5-\theta/\pi},$$

for parameter $-\pi/2 < \theta < \pi/2$ and $0 < u < 1$. Then the mean of U is $0.5 + \theta/\pi$ (returned as the fitted values) and the variance is $(\pi^2 - 4\theta^2)/(8\pi^2)$.

For both parameterizations Newton-Raphson is same as Fisher scoring.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Author(s)

T. W. Yee

References

Jorgensen, B. (1997). *The Theory of Dispersion Models*. London: Chapman & Hall.

Morris, C. N. (1982). Natural exponential families with quadratic variance functions. *The Annals of Statistics*, **10**(1), 65–80.

See Also

`gensh`, `extlogitlink`.

Examples

```
hdata <- data.frame(x2 = rnorm(nn <- 200))
hdata <- transform(hdata, y = rnorm(nn))  # Not very good data!
fit1 <- vglm(y ~ x2, hypersecant, hdata, trace = TRUE, crit = "c")
coef(fit1, matrix = TRUE)
fit1@misc$earg

# Not recommended:
fit2 <- vglm(y ~ x2, hypersecant(link = "identitylink"), hdata)
coef(fit2, matrix = TRUE)
fit2@misc$earg
```

`Hzeta` *Haight's Zeta Distribution*

Description

Density, distribution function, quantile function and random generation for Haight's zeta distribution with parameter shape.

Usage

```
dhzeta(x, shape, log = FALSE)
phzeta(q, shape, log.p = FALSE)
qhzeta(p, shape)
rhzeta(n, shape)
```

Arguments

`x, q, p, n`	Same meaning as `runif`.
`shape`	The positive shape parameter. Called α below.
`log,log.p`	Same meaning as in `pnorm` or `qnorm`.

Details

The probability function is

$$f(x) = (2x - 1)^{(-\alpha)} - (2x + 1)^{(-\alpha)},$$

where $\alpha > 0$ and $x = 1, 2, \ldots$.

Value

`dhzeta` gives the density, `phzeta` gives the distribution function, `qhzeta` gives the quantile function, and `rhzeta` generates random deviates.

Note

Given some response data, the **VGAM** family function `hzeta` estimates the parameter `shape`.

Author(s)

T. W. Yee and Kai Huang

See Also

`hzeta`, `zeta`, `zetaff`, `simulate.vlm`.

Examples

```
dhzeta(1:20, 2.1)
rhzeta(20, 2.1)

round(1000 * dhzeta(1:8, 2))
table(rhzeta(1000, 2))

## Not run:  shape <- 1.1; x <- 1:10
plot(x, dhzeta(x, shape = shape), type = "h", ylim = 0:1,
     sub = paste("shape =", shape), las = 1, col = "blue",
     ylab = "Probability", lwd = 2,
     main = "Haight's zeta: blue = density; orange = CDF")
lines(x+0.1, phzeta(x, shape = shape), col = "orange", lty = 3, lwd = 2,
      type = "h")

## End(Not run)
```

---

hzeta *Haight's Zeta Family Function*

Description

Estimating the parameter of Haight's zeta distribution

Usage

```
hzeta(lshape = "logloglink", ishape = NULL, nsimEIM = 100)
```

Arguments

`lshape`	Parameter link function for the parameter, called α below. See `Links` for more choices. Here, a log-log link keeps the parameter greater than one, meaning the mean is finite.
`ishape,nsimEIM`	See `CommonVGAMffArguments` for more information.

Details

The probability function is

$$f(y) = (2y - 1)^{(-\alpha)} - (2y + 1)^{(-\alpha)},$$

where the parameter $\alpha > 0$ and $y = 1, 2, \ldots$. The function `dhzeta` computes this probability function. The mean of Y, which is returned as fitted values, is $(1 - 2^{-\alpha})\zeta(\alpha)$ provided $\alpha > 1$, where ζ is Riemann's zeta function. The mean is a decreasing function of α. The mean is infinite if $\alpha \leq 1$, and the variance is infinite if $\alpha \leq 2$.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Author(s)

T. W. Yee

References

Johnson N. L., Kemp, A. W. and Kotz S. (2005). *Univariate Discrete Distributions*, 3rd edition, pp.533–4. Hoboken, New Jersey: Wiley.

See Also

Hzeta, zeta, zetaff, loglog, simulate.vlm.

Examples

```
shape <- exp(exp(-0.1))  # The parameter
hdata <- data.frame(y = rhzeta(n = 1000, shape))
fit <- vglm(y ~ 1, hzeta, data = hdata, trace = TRUE, crit = "coef")
coef(fit, matrix = TRUE)
Coef(fit)  # Useful for intercept-only models; should be same as shape
c(with(hdata, mean(y)), head(fitted(fit), 1))
summary(fit)
```

---

iam *Index from Array to Matrix*

Description

Maps the elements of an array containing symmetric positive-definite matrices to a matrix with sufficient columns to hold them (called matrix-band format.)

Usage

```
iam(j, k, M, both = FALSE, diag = TRUE)
```

Arguments

j	Usually an integer from the set {1:M} giving the row number of an element. However, the argument can also be a vector of length M, for selecting an entire row or column, e.g., iam(1:M, 1, M) or iam(1, 1:M, M).
k	An integer from the set {1:M} giving the column number of an element.
M	The number of linear/additive predictors. This is the dimension of each positive-definite symmetric matrix.
both	Logical. Return both the row and column indices? See below for more details.
diag	Logical. Return the indices for the diagonal elements? If FALSE then only the strictly upper triangular part of the matrix elements are used.

Details

Suppose we have n symmetric positive-definite square matrices, each M by M, and these are stored in an `array` of dimension `c(n,M,M)`. Then these can be more compactly represented by a `matrix` of dimension `c(n,K)` where `K` is an integer between `M` and `M*(M+1)/2` inclusive. The mapping between these two representations is given by this function. It firstly enumerates by the diagonal elements, followed by the band immediately above the diagonal, then the band above that one, etc. The last element is `(1,M)`. This function performs the mapping from elements `(j,k)` of symmetric positive-definite square matrices to the columns of another matrix representing such. This is called the *matrix-band* format and is used by the **VGAM** package.

Value

This function has a dual purpose depending on the value of `both`. If `both = FALSE` then the column number corresponding to the `j`-`k` element of the matrix is returned. If `both = TRUE` then `j` and `k` are ignored and a list with the following components are returned.

`row.index` The row indices of the upper triangular part of the matrix (This may or may not include the diagonal elements, depending on the argument `diagonal`).

`col.index` The column indices of the upper triangular part of the matrix (This may or may not include the diagonal elements, depending on the argument `diagonal`).

Note

This function is used in the `weight` slot of many **VGAM** family functions (see `vglmff-class`), especially those whose M is determined by the data, e.g., `dirichlet`, `multinomial`.

Author(s)

T. W. Yee

See Also

`vglmff-class`.

Examples

```
iam(1, 2, M = 3)  # The 4th coln represents elt (1,2) of a 3x3 matrix
iam(NULL, NULL, M = 3, both = TRUE)  # Return the row & column indices

dirichlet()@weight

M <- 4
temp1 <- iam(NA, NA, M = M, both = TRUE)
mat1 <- matrix(NA, M, M)
mat1[cbind(temp1$row, temp1$col)] = 1:length(temp1$row)
mat1  # More commonly used

temp2 <- iam(NA, NA, M = M, both = TRUE, diag = FALSE)
mat2 <- matrix(NA, M, M)
mat2[cbind(temp2$row, temp2$col)] = 1:length(temp2$row)
mat2  # Rarely used
```

`identitylink` *Identity Link Function*

Description

Computes the identity transformation, including its inverse and the first two derivatives.

Usage

```
   identitylink(theta, inverse = FALSE, deriv = 0, short = TRUE,
                tag = FALSE)
negidentitylink(theta, inverse = FALSE, deriv = 0, short = TRUE,
                tag = FALSE)
```

Arguments

`theta`	Numeric or character. See below for further details.
`inverse, deriv, short, tag`	Details at `Links`.

Details

The identity link function $g(\theta) = \theta$ should be available to every parameter estimated by the **VGAM** library. However, it usually results in numerical problems because the estimates lie outside the permitted range. Consequently, the result may contain `Inf`, `-Inf`, `NA` or `NaN`.

The function `negidentitylink` is the negative-identity link function and corresponds to $g(\theta) = -\theta$. This is useful for some models, e.g., in the literature supporting the `gevff` function it seems that half of the authors use $\xi = -k$ for the shape parameter and the other half use k instead of ξ.

Value

For `identitylink()`: for `deriv = 0`, the identity of `theta`, i.e., `theta` when `inverse = FALSE`, and if `inverse = TRUE` then `theta`. For `deriv = 1`, then the function returns *d* `eta` / *d* `theta` as a function of `theta` if `inverse = FALSE`, else if `inverse = TRUE` then it returns the reciprocal.

For `negidentitylink()`: the results are similar to `identitylink()` except for a sign change in most cases.

Author(s)

Thomas W. Yee

References

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*, 2nd ed. London: Chapman & Hall.

See Also

Links, loglink, logitlink, probitlink, powerlink.

Examples

```
identitylink((-5):5)
identitylink((-5):5, deriv = 1)
identitylink((-5):5, deriv = 2)
negidentitylink((-5):5)
negidentitylink((-5):5, deriv = 1)
negidentitylink((-5):5, deriv = 2)
```

Influence — *Influence Function (S4 generic) of a Fitted Model*

Description

Returns a matrix containing the influence function of a fitted model, e.g., a "vglm" object.

Usage

```
Influence(object, ...)
Influence.vglm(object, weighted = TRUE, ...)
```

Arguments

`object`	an object, especially that of class "vglm"—see vglm-class. Currently other classes such as "vgam" are not yet implemented.
`weighted`	Logical. Include the prior weights? Currently only `TRUE` is accepted. This might change in the future and/or the default value might change.
`...`	any additional arguments such as to allow or disallow the prior weights.

Details

Influence functions are useful in fields such as sample survey theory, e.g., **survey**, **svyVGAM**. For each $i = 1, \ldots, n$, the formula is approximately $-IU$ where I is the weighted Fisher information matrix and U is the ith score vector.

Value

An `n` by `p.vlm` matrix.

Warning

This function is currently experimental and defaults may change. Use with caution! The functions here should not be confused with lm.influence.

See Also

vglm, vglm-class, **survey**.

Examples

```
pneumo <- transform(pneumo, let = log(exposure.time))
fit <- vglm(cbind(normal, mild, severe) ~ let, acat, data = pneumo)
coef(fit)  # 8-vector
Influence(fit)  # 8 x 4
all(abs(colSums(Influence(fit))) < 1e-6)  # TRUE
```

---

inv.binomial *Inverse Binomial Distribution Family Function*

Description

Estimates the two parameters of an inverse binomial distribution by maximum likelihood estimation.

Usage

```
inv.binomial(lrho = extlogitlink(min = 0.5, max = 1),
    llambda = "loglink", irho = NULL, ilambda = NULL, zero = NULL)
```

Arguments

lrho, llambda	Link function for the ρ and λ parameters. See Links for more choices.
irho, ilambda	Numeric. Optional initial values for ρ and λ.
zero	See CommonVGAMffArguments.

Details

The inverse binomial distribution of Yanagimoto (1989) has density function

$$f(y;\rho,\lambda) = \frac{\lambda\,\Gamma(2y+\lambda)}{\Gamma(y+1)\,\Gamma(y+\lambda+1)}\{\rho(1-\rho)\}^y\rho^\lambda$$

where $y = 0, 1, 2, \ldots$ and $\frac{1}{2} < \rho < 1$, and $\lambda > 0$. The first two moments exist for $\rho > \frac{1}{2}$; then the mean is $\lambda(1-\rho)/(2\rho-1)$ (returned as the fitted values) and the variance is $\lambda\rho(1-\rho)/(2\rho-1)^3$. The inverse binomial distribution is a special case of the generalized negative binomial distribution of Jain and Consul (1971). It holds that $Var(Y) > E(Y)$ so that the inverse binomial distribution is overdispersed compared with the Poisson distribution.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm and vgam.

Note

This **VGAM** family function only works reasonably well with intercept-only models. Good initial values are needed; if convergence failure occurs use `irho` and/or `ilambda`.

Some elements of the working weight matrices use the expected information matrix while other elements use the observed information matrix. Yet to do: using the mean and the reciprocal of λ results in an EIM that is diagonal.

Author(s)

T. W. Yee

References

Yanagimoto, T. (1989). The inverse binomial distribution as a statistical model. *Communications in Statistics: Theory and Methods*, **18**, 3625–3633.

Jain, G. C. and Consul, P. C. (1971). A generalized negative binomial distribution. *SIAM Journal on Applied Mathematics*, **21**, 501–513.

Jorgensen, B. (1997). *The Theory of Dispersion Models*. London: Chapman & Hall

See Also

`negbinomial`, `poissonff`.

Examples

```
idata <- data.frame(y = rnbinom(n <- 1000, mu = exp(3), size = exp(1)))
fit <- vglm(y ~ 1, inv.binomial, data = idata, trace = TRUE)
with(idata, c(mean(y), head(fitted(fit), 1)))
summary(fit)
coef(fit, matrix = TRUE)
Coef(fit)
sum(weights(fit))  # Sum of the prior weights
sum(weights(fit, type = "work"))  # Sum of the working weights
```

`Inv.gaussian` *The Inverse Gaussian Distribution*

Description

Density, distribution function and random generation for the inverse Gaussian distribution.

Usage

```
dinv.gaussian(x, mu, lambda, log = FALSE)
pinv.gaussian(q, mu, lambda)
rinv.gaussian(n, mu, lambda)
```

Arguments

x, q	vector of quantiles.
n	number of observations. If length(n) > 1 then the length is taken to be the number required.
mu	the mean parameter.
lambda	the λ parameter.
log	Logical. If log = TRUE then the logarithm of the density is returned.

Details

See inv.gaussianff, the **VGAM** family function for estimating both parameters by maximum likelihood estimation, for the formula of the probability density function.

Value

dinv.gaussian gives the density, pinv.gaussian gives the distribution function, and rinv.gaussian generates random deviates.

Note

Currently qinv.gaussian is unavailable.

Author(s)

T. W. Yee

References

Johnson, N. L. and Kotz, S. and Balakrishnan, N. (1994). *Continuous Univariate Distributions*, 2nd edition, Volume 1, New York: Wiley.

Taraldsen, G. and Lindqvist, B. H. (2005). The multiple roots simulation algorithm, the inverse Gaussian distribution, and the sufficient conditional Monte Carlo method. *Preprint Statistics No. 4/2005*, Norwegian University of Science and Technology, Trondheim, Norway.

See Also

inv.gaussianff, waldff.

Examples

```
## Not run:  x <- seq(-0.05, 4, len = 300)
plot(x, dinv.gaussian(x, mu = 1, lambda = 1), type = "l",
     col = "blue",las = 1, main =
     "blue is density, orange is cumulative distribution function")
abline(h = 0, col = "gray", lty = 2)
lines(x, pinv.gaussian(x, mu = 1, lambda = 1), type = "l", col = "orange")
## End(Not run)
```

inv.gaussianff *Inverse Gaussian Distribution Family Function*

Description

Estimates the two parameters of the inverse Gaussian distribution by maximum likelihood estimation.

Usage

```
inv.gaussianff(lmu = "loglink", llambda = "loglink",
      imethod = 1, ilambda = NULL,
      parallel = FALSE, ishrinkage = 0.99, zero = NULL)
```

Arguments

`lmu, llambda` Parameter link functions for the μ and λ parameters. See `Links` for more choices.

`ilambda, parallel` See `CommonVGAMffArguments` for more information. If `parallel = TRUE` then the constraint is not applied to the intercept.

`imethod, ishrinkage, zero` See `CommonVGAMffArguments` for information.

Details

The standard ("canonical") form of the inverse Gaussian distribution has a density that can be written as

$$f(y; \mu, \lambda) = \sqrt{\lambda/(2\pi y^3)} \exp\left(-\lambda(y-\mu)^2/(2y\mu^2)\right)$$

where $y > 0$, $\mu > 0$, and $\lambda > 0$. The mean of Y is μ and its variance is μ^3/λ. By default, $\eta_1 = \log(\mu)$ and $\eta_2 = \log(\lambda)$. The mean is returned as the fitted values. This **VGAM** family function can handle multiple responses (inputted as a matrix).

Value

An object of class "vglmff" (see `vglmff-class`). The object is used by modelling functions such as `vglm`, `rrvglm` and `vgam`.

Note

The inverse Gaussian distribution can be fitted (to a certain extent) using the usual GLM framework involving a scale parameter. This family function is different from that approach in that it estimates both parameters by full maximum likelihood estimation.

Author(s)

T. W. Yee

References

Johnson, N. L. and Kotz, S. and Balakrishnan, N. (1994). *Continuous Univariate Distributions*, 2nd edition, Volume 1, New York: Wiley.

Forbes, C., Evans, M., Hastings, N. and Peacock, B. (2011). *Statistical Distributions*, Hoboken, NJ, USA: John Wiley and Sons, Fourth edition.

See Also

`Inv.gaussian`, `waldff`, `bisa`.

The R package **SuppDists** has several functions for evaluating the density, distribution function, quantile function and generating random numbers from the inverse Gaussian distribution.

Examples

```
idata <- data.frame(x2 = runif(nn <- 1000))
idata <- transform(idata, mymu   = exp(2 + 1 * x2),
                          Lambda = exp(2 + 1 * x2))
idata <- transform(idata, y = rinv.gaussian(nn, mu = mymu, Lambda))
fit1 <-   vglm(y ~ x2, inv.gaussianff, data = idata, trace = TRUE)
rrig <- rrvglm(y ~ x2, inv.gaussianff, data = idata, trace = TRUE)
coef(fit1, matrix = TRUE)
coef(rrig, matrix = TRUE)
Coef(rrig)
summary(fit1)
```

`Inv.lomax` *The Inverse Lomax Distribution*

Description

Density, distribution function, quantile function and random generation for the inverse Lomax distribution with shape parameter `p` and scale parameter `scale`.

Usage

```
dinv.lomax(x, scale = 1, shape2.p, log = FALSE)
pinv.lomax(q, scale = 1, shape2.p, lower.tail = TRUE, log.p = FALSE)
qinv.lomax(p, scale = 1, shape2.p, lower.tail = TRUE, log.p = FALSE)
rinv.lomax(n, scale = 1, shape2.p)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. If `length(n)` > 1, the length is taken to be the number required.

`shape2.p` shape parameter.

`scale` scale parameter.

`log` Logical. If `log = TRUE` then the logarithm of the density is returned.

`lower.tail, log.p`
Same meaning as in `pnorm` or `qnorm`.

Details

See `inv.lomax`, which is the **VGAM** family function for estimating the parameters by maximum likelihood estimation.

Value

`dinv.lomax` gives the density, `pinv.lomax` gives the distribution function, `qinv.lomax` gives the quantile function, and `rinv.lomax` generates random deviates.

Note

The inverse Lomax distribution is a special case of the 4-parameter generalized beta II distribution.

Author(s)

T. W. Yee

References

Kleiber, C. and Kotz, S. (2003). *Statistical Size Distributions in Economics and Actuarial Sciences*, Hoboken, NJ, USA: Wiley-Interscience.

See Also

`inv.lomax`, `genbetaII`.

Examples

```
idata <- data.frame(y = rinv.lomax(n = 1000, exp(2), exp(1)))
fit <- vglm(y ~ 1, inv.lomax, idata, trace = TRUE, crit = "coef")
coef(fit, matrix = TRUE)
Coef(fit)
```

inv.lomax *Inverse Lomax Distribution Family Function*

Description

Maximum likelihood estimation of the 2-parameter inverse Lomax distribution.

Usage

```
inv.lomax(lscale = "loglink", lshape2.p = "loglink", iscale = NULL,
    ishape2.p = NULL, imethod = 1, gscale = exp(-5:5),
    gshape2.p = exp(-5:5), probs.y = c(0.25, 0.5, 0.75),
    zero = "shape2.p")
```

Arguments

`lscale, lshape2.p`
: Parameter link functions applied to the (positive) parameters b, and p. See `Links` for more choices.

`iscale, ishape2.p, imethod, zero`
: See `CommonVGAMffArguments` for information. For `imethod = 2` a good initial value for `ishape2.p` is needed to obtain a good estimate for the other parameter.

`gscale, gshape2.p`
: See `CommonVGAMffArguments` for information.

`probs.y`
: See `CommonVGAMffArguments` for information.

Details

The 2-parameter inverse Lomax distribution is the 4-parameter generalized beta II distribution with shape parameters $a = q = 1$. It is also the 3-parameter Dagum distribution with shape parameter $a = 1$, as well as the beta distribution of the second kind with $q = 1$. More details can be found in Kleiber and Kotz (2003).

The inverse Lomax distribution has density

$$f(y) = py^{p-1}/[b^p\{1 + y/b\}^{p+1}]$$

for $b > 0$, $p > 0$, $y \geq 0$. Here, b is the scale parameter `scale`, and `p` is a shape parameter. The mean does not seem to exist; the *median* is returned as the fitted values. This family function handles multiple responses.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Note

See the notes in `genbetaII`.

Author(s)

T. W. Yee

References

Kleiber, C. and Kotz, S. (2003). *Statistical Size Distributions in Economics and Actuarial Sciences*, Hoboken, NJ, USA: Wiley-Interscience.

See Also

`inv.lomax`, `genbetaII`, `betaII`, `dagum`, `sinmad`, `fisk`, `lomax`, `paralogistic`, `inv.paralogistic`, `simulate.vlm`.

Examples

```
idata <- data.frame(y = rinv.lomax(2000, sc = exp(2), exp(1)))
fit <- vglm(y ~ 1, inv.lomax, data = idata, trace = TRUE)
fit <- vglm(y ~ 1, inv.lomax(iscale = exp(3)), data = idata,
            trace = TRUE, epsilon = 1e-8, crit = "coef")
coef(fit, matrix = TRUE)
Coef(fit)
summary(fit)
```

`Inv.paralogistic` *The Inverse Paralogistic Distribution*

Description

Density, distribution function, quantile function and random generation for the inverse paralogistic distribution with shape parameters `a` and `p`, and scale parameter `scale`.

Usage

```
dinv.paralogistic(x, scale = 1, shape1.a, log = FALSE)
pinv.paralogistic(q, scale = 1, shape1.a, lower.tail = TRUE,
                  log.p = FALSE)
qinv.paralogistic(p, scale = 1, shape1.a, lower.tail = TRUE,
                  log.p = FALSE)
rinv.paralogistic(n, scale = 1, shape1.a)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. If `length(n)` > 1, the length is taken to be the number required.
`shape1.a`	shape parameter.

scale — scale parameter.

log — Logical. If `log = TRUE` then the logarithm of the density is returned.

lower.tail, log.p
Same meaning as in `pnorm` or `qnorm`.

Details

See `inv.paralogistic`, which is the **VGAM** family function for estimating the parameters by maximum likelihood estimation.

Value

`dinv.paralogistic` gives the density, `pinv.paralogistic` gives the distribution function, `qinv.paralogistic` gives the quantile function, and `rinv.paralogistic` generates random deviates.

Note

The inverse paralogistic distribution is a special case of the 4-parameter generalized beta II distribution.

Author(s)

T. W. Yee

References

Kleiber, C. and Kotz, S. (2003). *Statistical Size Distributions in Economics and Actuarial Sciences*, Hoboken, NJ, USA: Wiley-Interscience.

See Also

`inv.paralogistic`, `genbetaII`.

Examples

```
idata <- data.frame(y = rinv.paralogistic(3000, exp(1), sc = exp(2)))
fit <- vglm(y ~ 1, inv.paralogistic(lss = FALSE, ishape1.a = 2.1),
            data = idata, trace = TRUE, crit = "coef")
coef(fit, matrix = TRUE)
Coef(fit)
```

inv.paralogistic *Inverse Paralogistic Distribution Family Function*

Description

Maximum likelihood estimation of the 2-parameter inverse paralogistic distribution.

Usage

```
inv.paralogistic(lscale = "loglink", lshape1.a = "loglink",
    iscale = NULL, ishape1.a = NULL, imethod = 1,
    lss = TRUE, gscale = exp(-5:5),
    gshape1.a = seq(0.75, 4, by = 0.25), probs.y = c(0.25, 0.5,
    0.75), zero = "shape")
```

Arguments

`lss` See `CommonVGAMffArguments` for important information.

`lshape1.a, lscale`
: Parameter link functions applied to the (positive) parameters `a` and `scale`. See `Links` for more choices.

`iscale, ishape1.a, imethod, zero`
: See `CommonVGAMffArguments` for information. For `imethod = 2` a good initial value for `ishape1.a` is needed to obtain a good estimate for the other parameter.

`gscale, gshape1.a`
: See `CommonVGAMffArguments` for information.

`probs.y` See `CommonVGAMffArguments` for information.

Details

The 2-parameter inverse paralogistic distribution is the 4-parameter generalized beta II distribution with shape parameter $q = 1$ and $a = p$. It is the 3-parameter Dagum distribution with $a = p$. More details can be found in Kleiber and Kotz (2003).

The inverse paralogistic distribution has density

$$f(y) = a^2 y^{a^2-1} / [b^{a^2} \{1 + (y/b)^a\}^{a+1}]$$

for $a > 0$, $b > 0$, $y \geq 0$. Here, b is the scale parameter `scale`, and a is the shape parameter. The mean is

$$E(Y) = b \, \Gamma(a + 1/a) \, \Gamma(1 - 1/a) / \Gamma(a)$$

provided $a > 1$; these are returned as the fitted values. This family function handles multiple responses.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Note

See the notes in genbetaII.

Author(s)

T. W. Yee

References

Kleiber, C. and Kotz, S. (2003). *Statistical Size Distributions in Economics and Actuarial Sciences*, Hoboken, NJ, USA: Wiley-Interscience.

See Also

Inv.paralogistic, genbetaII, betaII, dagum, sinmad, fisk, inv.lomax, lomax, paralogistic, simulate.vlm.

Examples

```
idata <- data.frame(y = rinv.paralogistic(3000, exp(1), sc = exp(2)))
fit <- vglm(y ~ 1, inv.paralogistic(lss = FALSE), idata, trace = TRUE)
fit <- vglm(y ~ 1, inv.paralogistic(imethod = 2, ishape1.a = 4),
            data = idata, trace = TRUE, crit = "coef")
coef(fit, matrix = TRUE)
Coef(fit)
summary(fit)
```

`is.buggy` *Does the Fitted Object Suffer from a Known Bug?*

Description

Checks to see if a fitted object suffers from some known bug.

Usage

```
is.buggy(object, ...)
is.buggy.vlm(object, each.term = FALSE, ...)
```

Arguments

object	A fitted **VGAM** object, e.g., from vgam.
each.term	Logical. If TRUE then a logical is returned for each term.
...	Unused for now.

Details

It is known that vgam with s terms do not correctly handle constraint matrices (cmat, say) when crossprod(cmat) is not diagonal. This function detects whether this is so or not. Note that probably all **VGAM** family functions have defaults where all crossprod(cmat)s are diagonal, therefore do not suffer from this bug. It is more likely to occur if the user inputs constraint matrices using the constraints argument (and setting zero = NULL if necessary).

Second-generation VGAMs based on sm.ps are a modern alternative to using s. It does not suffer from this bug. However, G2-VGAMs require a reasonably large sample size in order to work more reliably.

Value

The default is a single logical (TRUE if any term is TRUE), otherwise a vector of such with each element corresponding to a term. If the value is TRUE then I suggest replacing the VGAM by a similar model fitted by vglm and using regression splines, e.g., bs, ns.

Note

When the bug is fixed this function may be withdrawn, otherwise always return FALSEs!

Author(s)

T. W. Yee

See Also

vgam. vglm, s, sm.ps, bs, ns.

Examples

```
fit1 <- vgam(cbind(agaaus, kniexc) ~ s(altitude, df = c(3, 4)),
             binomialff(multiple.responses = TRUE), data = hunua)
is.buggy(fit1)  # Okay
is.buggy(fit1, each.term = TRUE)  # No terms are buggy
fit2 <-
  vgam(cbind(agaaus, kniexc) ~ s(altitude, df = c(3, 4)),
       binomialff(multiple.responses = TRUE), data = hunua,
       constraints =
       list("(Intercept)" = diag(2),
            "s(altitude, df = c(3, 4))" = matrix(c(1, 1, 0, 1), 2, 2)))
is.buggy(fit2)  # TRUE
is.buggy(fit2, each.term = TRUE)
constraints(fit2)

# fit2b is an approximate alternative to fit2:
fit2b <-
  vglm(cbind(agaaus, kniexc) ~ bs(altitude, df=3) + bs(altitude, df=4),
       binomialff(multiple.responses = TRUE), data = hunua,
       constraints =
         list("(Intercept)" = diag(2),
```

```
                "bs(altitude, df = 3)" = rbind(1, 1),
                "bs(altitude, df = 4)" = rbind(0, 1)))
is.buggy(fit2b)  # Okay
is.buggy(fit2b, each.term = TRUE)
constraints(fit2b)
```

`is.crossing` *Quantile Crossing Detection*

Description

Returns a logical from testing whether an object such as an extlogF1() VGLM object has crossing quantiles.

Usage

```
is.crossing.vglm(object, ...)
```

Arguments

`object`	an object such as a `vglm` object with family function `extlogF1`.
`...`	additional optional arguments. Currently unused.

Details

This function was specifically written for a `vglm` with family function `extlogF1`. It examines the fitted quantiles to see if any cross. Note that if one uses regression splines such as `bs` and `ns` then it is possible that they cross at values of the covariate space that are not represented by actual data. One could use linear interpolation between fitted values to get around this problem.

Value

A logical. If `TRUE` then one can try fit a similar model by combining columns of the constraint matrices so that crossing no longer holds; see `fix.crossing`. For LMS-Box-Cox type quantile regression models it is impossible for the quantiles to cross, by definition, hence `FALSE` is returned; see `lms.bcn`.

See Also

`extlogF1`, `fix.crossing`, `lms.bcn`. `vglm`.

Examples

```
## Not run:  ooo <- with(bmi.nz, order(age))
bmi.nz <- bmi.nz[ooo, ]  # Sort by age
with(bmi.nz, plot(age, BMI, col = "blue"))
mytau <- c(50, 93, 95, 97) / 100  # Some quantiles are quite close
fit1 <- vglm(BMI ~ ns(age, 7), extlogF1(mytau), bmi.nz, trace = TRUE)
plot(BMI ~ age, bmi.nz, col = "blue", las = 1,
     main = "Partially parallel (darkgreen) & nonparallel quantiles",
     sub = "Crossing quantiles are orange")
is.crossing(fit1)
matlines(with(bmi.nz, age), fitted(fit1), lty = 1, col = "orange")
## End(Not run)
```

---

is.parallel *Parallelism Constraint Matrices*

Description

Returns a logical vector from a test of whether an object such as a matrix or VGLM object corresponds to a parallelism assumption.

Usage

```
is.parallel.matrix(object, ...)
is.parallel.vglm(object, type = c("term", "lm"), ...)
```

Arguments

object	an object such as a constraint matrix or a vglm object.
type	passed into constraints.
...	additional optional arguments. Currently unused.

Details

These functions may be useful for categorical models such as propodds, cumulative, acat, cratio, sratio, multinomial.

Value

A vector of logicals, testing whether each constraint matrix is a one-column matrix of ones. Note that parallelism can still be thought of as holding if the constraint matrix has a non-zero but constant values, however, this is currently not implemented. No checking is done that the constraint matrices have the same number of rows.

See Also

constraints, vglm.

Examples

```
## Not run:  require("VGAMdata")
fit <- vglm(educ ~ sm.bs(age) * sex + ethnicity,
            cumulative(parallel = TRUE), head(xs.nz, 200))
is.parallel(fit)
is.parallel(fit, type = "lm")  # For each column of the LM matrix

## End(Not run)
```

---

is.smart *Test For a Smart Object*

Description

Tests an object to see if it is smart.

Usage

```
is.smart(object)
```

Arguments

object	a function or a fitted model.

Details

If `object` is a function then this function looks to see whether `object` has the logical attribute `"smart"`. If so then this is returned, else `FALSE`.

If `object` is a fitted model then this function looks to see whether `object@smart.prediction` or `object\$smart.prediction` exists. If it does and it is not equal to `list(smart.arg=FALSE)` then a `TRUE` is returned, else `FALSE`. The reason for this is because, e.g., `lm(...,smart=FALSE)` and `vglm(...,smart=FALSE)`, will return such a specific list.

Writers of smart functions manually have to assign this attribute to their smart function after it has been written.

Value

Returns `TRUE` or `FALSE`, according to whether the `object` is smart or not.

Examples

```
is.smart(sm.min1)  # TRUE
is.smart(sm.poly)  # TRUE
library(splines)
is.smart(sm.bs)  # TRUE
is.smart(sm.ns)  # TRUE
is.smart(tan)  # FALSE
## Not run:
```

```
udata <- data.frame(x2 = rnorm(9))
fit1 <- vglm(rnorm(9) ~ x2, uninormal, data = udata)
is.smart(fit1)  # TRUE
fit2 <- vglm(rnorm(9) ~ x2, uninormal, data = udata, smart = FALSE)
is.smart(fit2)  # FALSE
fit2@smart.prediction

## End(Not run)
```

`is.zero` *Zero Constraint Matrices*

Description

Returns a logical vector from a test of whether an object such as a matrix or VGLM object corresponds to a 'zero' assumption.

Usage

```
is.zero.matrix(object, ...)
is.zero.vglm(object, ...)
```

Arguments

`object`	an object such as a coefficient matrix of a `vglm` object, or a `vglm` object.
`...`	additional optional arguments. Currently unused.

Details

These functions test the effect of the `zero` argument on a `vglm` object or the coefficient matrix of a `vglm` object. The latter is obtained by `coef(vglmObject, matrix = TRUE)`.

Value

A vector of logicals, testing whether each linear/additive predictor has the `zero` argument applied to it. It is `TRUE` if that linear/additive predictor is intercept-only, i.e., all other regression coefficients are set to zero.

No checking is done for the intercept term at all, i.e., that it was estimated in the first place.

See Also

`constraints`, `vglm`, `CommonVGAMffArguments`.

Examples

```
coalminers <- transform(coalminers, Age = (age - 42) / 5)
fit <- vglm(cbind(nBnW,nBW,BnW,BW) ~ Age, binom2.or(zero = NULL),
            data = coalminers)
is.zero(fit)
is.zero(coef(fit, matrix = TRUE))
```

kendall.tau *Kendall's Tau Statistic*

Description

Computes Kendall's Tau, which is a rank-based correlation measure, between two vectors.

Usage

```
kendall.tau(x, y, exact = FALSE, max.n = 3000)
```

Arguments

`x, y`	Numeric vectors. Must be of equal length. Ideally their values are continuous and not too discrete. Let `length(x)` be N, say.
`exact`	Logical. If `TRUE` then the exact value is computed.
`max.n`	Numeric. If `exact = FALSE` and `length(x)` is more than `max.n` then a random sample of `max.n` pairs are chosen.

Details

Kendall's tau is a measure of dependency in a bivariate distribution. Loosely, two random variables are *concordant* if large values of one random variable are associated with large values of the other random variable. Similarly, two random variables are *disconcordant* if large values of one random variable are associated with small values of the other random variable. More formally, if `(x[i] - x[j])*(y[i] - y[j]) > 0` then that comparison is concordant $(i \neq j)$. And if `(x[i] - x[j])*(y[i] - y[j]) < 0` then that comparison is disconcordant $(i \neq j)$. Out of `choose(N, 2)` comparisons, let c and d be the number of concordant and disconcordant pairs. Then Kendall's tau can be estimated by $(c-d)/(c+d)$. If there are ties then half the ties are deemed concordant and half disconcordant so that $(c-d)/(c+d+t)$ is used.

Value

Kendall's tau, which lies between -1 and 1.

Warning

If `length(x)` is large then the cost is $O(N^2)$, which is expensive! Under these circumstances it is not advisable to set `exact = TRUE` or `max.n` to a very large number.

See Also

`binormalcop`, `cor`.

Examples

```
N <- 5000; x <- 1:N; y <- runif(N)
true.rho <- -0.8
ymat <- rbinorm(N, cov12 =  true.rho)  # Bivariate normal, aka N_2
x <- ymat[, 1]
y <- ymat[, 2]

## Not run: plot(x, y, col = "blue")

kendall.tau(x, y)  # A random sample is taken here
kendall.tau(x, y)  # A random sample is taken here

kendall.tau(x, y, exact = TRUE)  # Costly if length(x) is large
kendall.tau(x, y, max.n = N)     # Same as exact = TRUE

(rhohat <- sin(kendall.tau(x, y) * pi / 2))  # Holds for N_2 actually
true.rho  # rhohat should be near this value
```

KLD *Kullback-Leibler Divergence*

Description

Calculates the Kullback-Leibler divergence for certain fitted model objects

Usage

```
KLD(object, ...)
KLDvglm(object, ...)
```

Arguments

`object`	Some **VGAM** object, for example, having class `vglm-class`. Currently `object` must be intercept-only.
`...`	Other possible arguments fed into `KLDvglm` in order to compute the KLD.

Details

The *Kullback-Leibler divergence* (KLD), or *relative entropy*, is a measure of how one probability distribution differs from a second reference probability distribution. Currently the **VGAM** package computes the KLD for GAITD regression models (e.g., see `gaitdpoisson` and `gaitdnbinomial`) where the reference distribution is the (unscaled) parent or base distribution. For such, the formula for the KLD simplifies somewhat. Hence one can obtain a quantitative measure for the overall effect of altering, inflating, truncating and deflating certain (special) values.

Value

Returns a numeric nonnegative value with the corresponding KLD. A 0 value means no difference between an ordinary parent or base distribution.

Warning

Numerical problems might occur if any of the evaluated probabilities of the unscaled parent distribution are very close to 0.

Author(s)

T. W. Yee.

References

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *Annals of Mathematical Statistics*, **22**, 79–86.

M'Kendrick, A. G. (1925). Applications of mathematics to medical problems. *Proc. Edinb. Math. Soc.*, **44**, 98–130.

See Also

gaitdpoisson, gaitdnbinomial.

Examples

```
#  McKendrick (1925): Data from 223 Indian village households
cholera <- data.frame(ncases = 0:4,  # Number of cholera cases,
                      wfreq  = c(168, 32, 16, 6, 1))  # Frequencies
fit7 <- vglm(ncases ~ 1, gaitdpoisson(i.mlm = 0, ilambda.p = 1),
             weight = wfreq, data = cholera, trace = TRUE)
coef(fit7, matrix = TRUE)
KLD(fit7)
```

Kumar *The Kumaraswamy Distribution*

Description

Density, distribution function, quantile function and random generation for the Kumaraswamy distribution.

Usage

```
dkumar(x, shape1, shape2, log = FALSE)
pkumar(q, shape1, shape2, lower.tail = TRUE, log.p = FALSE)
qkumar(p, shape1, shape2, lower.tail = TRUE, log.p = FALSE)
rkumar(n, shape1, shape2)
```

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
n	number of observations. If length(n) > 1 then the length is taken to be the number required.
shape1, shape2	positive shape parameters.
log	Logical. If log = TRUE then the logarithm of the density is returned.
lower.tail, log.p	Same meaning as in pnorm or qnorm.

Details

See kumar, the **VGAM** family function for estimating the parameters, for the formula of the probability density function and other details.

Value

dkumar gives the density, pkumar gives the distribution function, qkumar gives the quantile function, and rkumar generates random deviates.

Author(s)

T. W. Yee and Kai Huang

See Also

kumar.

Examples

```
## Not run:
shape1 <- 2; shape2 <- 2; nn <- 201; # shape1 <- shape2 <- 0.5;
x <- seq(-0.05, 1.05, len = nn)
plot(x, dkumar(x, shape1, shape2), type = "l", las = 1,
     ylab = paste("dkumar(shape1 = ", shape1,
                  ", shape2 = ", shape2, ")"),
     col = "blue", cex.main = 0.8, ylim = c(0,1.5),
     main = "Blue is density, orange is the CDF",
     sub = "Red lines are the 10,20,...,90 percentiles")
lines(x, pkumar(x, shape1, shape2), col = "orange")
probs <- seq(0.1, 0.9, by = 0.1)
Q <- qkumar(probs, shape1, shape2)
lines(Q, dkumar(Q, shape1, shape2), col = "red", lty = 3, type = "h")
lines(Q, pkumar(Q, shape1, shape2), col = "red", lty = 3, type = "h")
abline(h = probs, col = "red", lty = 3)
max(abs(pkumar(Q, shape1, shape2) - probs))  # Should be 0

## End(Not run)
```

kumar *Kumaraswamy Regression Family Function*

Description

Estimates the two parameters of the Kumaraswamy distribution by maximum likelihood estimation.

Usage

```
kumar(lshape1 = "loglink", lshape2 = "loglink",
      ishape1 = NULL,   ishape2 = NULL,
      gshape1 = exp(2*ppoints(5) - 1), tol12 = 1.0e-4, zero = NULL)
```

Arguments

`lshape1, lshape2`
: Link function for the two positive shape parameters, respectively, called a and b below. See `Links` for more choices.

`ishape1, ishape2`
: Numeric. Optional initial values for the two positive shape parameters.

`tol12`
: Numeric and positive. Tolerance for testing whether the second shape parameter is either 1 or 2. If so then the working weights need to handle these singularities.

`gshape1`
: Values for a grid search for the first shape parameter. See `CommonVGAMffArguments` for more information.

`zero`
: See `CommonVGAMffArguments`.

Details

The Kumaraswamy distribution has density function

$$f(y; a = shape1, b = shape2) = aby^{a-1}(1 - y^a)^{b-1}$$

where $0 < y < 1$ and the two shape parameters, a and b, are positive. The mean is $b \times Beta(1 + 1/a, b)$ (returned as the fitted values) and the variance is $b \times Beta(1 + 2/a, b) - (b \times Beta(1 + 1/a, b))^2$. Applications of the Kumaraswamy distribution include the storage volume of a water reservoir. Fisher scoring is implemented. Handles multiple responses (matrix input).

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Author(s)

T. W. Yee

References

Kumaraswamy, P. (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, **46**, 79–88.

Jones, M. C. (2009). Kumaraswamy's distribution: A beta-type distribution with some tractability advantages. *Statistical Methodology*, **6**, 70–81.

See Also

dkumar, betaff, simulate.vlm.

Examples

```
shape1 <- exp(1); shape2 <- exp(2)
kdata <- data.frame(y = rkumar(n = 1000, shape1, shape2))
fit <- vglm(y ~ 1, kumar, data = kdata, trace = TRUE)
c(with(kdata, mean(y)), head(fitted(fit), 1))
coef(fit, matrix = TRUE)
Coef(fit)
summary(fit)
```

---

lakeO — *Annual catches on Lake Otamangakau from October 1974 to October 1989*

Description

Rainbow and brown trout catches by a Mr Swainson at Lake Otamangakau in the central North Island of New Zealand during the 1970s and 1980s.

Usage

```
data(lakeO)
```

Format

A data frame with 15 observations on the following 5 variables.

`year` a numeric vector, the season began on 1 October of the year and ended 12 months later.

`total.fish` a numeric vector, the total number of fish caught during the season. Simply the sum of brown and rainbow trout.

`brown` a numeric vector, the number of brown trout (*Salmo trutta*) caught.

`rainbow` a numeric vector, the number of rainbow trout (*Oncorhynchus mykiss*) caught.

`visits` a numeric vector, the number of visits during the season that the angler made to the lake. It is necessary to assume that the visits were of an equal time length in order to interpret the usual Poisson regressions.

Details

The data was extracted from the season summaries at Lake Otamangakau by Anthony Swainson for the seasons 1974–75 to 1988–89.

Mr Swainson was one of a small group of regular fly fishing anglers and kept a diary of his catches. Lake Otamangakau is a lake of area 1.8 squared km and has a maximum depth of about 12m, and is located in the central North Island of New Zealand. It is trout-infested and known for its trophy-sized fish.

See also `trap0`.

Source

Table 7.2 of the reference below. Thanks to Dr Michel Dedual for a copy of the report and for help reading the final year's data. The report is available from TWY on request.

References

Dedual, M. and MacLean, G. and Rowe, D. and Cudby, E., *The Trout Population and Fishery of Lake Otamangakau—Interim Report*. National Institute of Water and Atmospheric Research, Hamilton, New Zealand. Consultancy Report Project No. ELE70207, (Dec 1996).

Examples

```
data(lake0)
lake0
summary(lake0)
```

`lambertW` *The Lambert W Function*

Description

Computes the Lambert *W* function for real values.

Usage

```
lambertW(x, tolerance = 1e-10, maxit = 50)
```

Arguments

`x`	A vector of reals.
`tolerance`	Accuracy desired.
`maxit`	Maximum number of iterations of third-order Halley's method.

Details

The Lambert W function is the root of the equation $W(z)\exp(W(z)) = z$ for complex z. If z is real and $-1/e < z < 0$ then it has two possible real values, and currently only the upper branch (often called W_0) is computed so that a value that is ≥ -1 is returned.

Value

This function returns the principal branch of the W function for *real* z. It returns $W(z) \geq -1$, and `NA` for $z < -1/e$.

Note

If convergence does not occur then increase the value of `maxit` and/or `tolerance`.

Yet to do: add an argument `lbranch = TRUE` to return the lower branch (often called W_{-1}) for real $-1/e \leq z < 0$; this would give $W(z) \leq -1$.

Author(s)

T. W. Yee

References

Corless, R. M. and Gonnet, G. H. and Hare, D. E. G. and Jeffrey, D. J. and Knuth, D. E. (1996). On the Lambert W function. *Advances in Computational Mathematics*, **5**(4), 329–359.

See Also

`log`, `exp`, `bell`. There is also a package called **LambertW**.

Examples

```
 ## Not run:
curve(lambertW, -exp(-1), 3, xlim = c(-1, 3), ylim = c(-2, 1),
      las = 1, col = "orange", n = 1001)
abline(v = -exp(-1), h = -1, lwd = 2, lty = "dotted", col = "gray")
abline(h = 0, v = 0, lty = "dashed", col = "blue")
## End(Not run)
```

`laplace` *Laplace Regression Family Function*

Description

Maximum likelihood estimation of the 2-parameter classical Laplace distribution.

Usage

```
laplace(llocation = "identitylink", lscale = "loglink",
  ilocation = NULL, iscale = NULL, imethod = 1, zero = "scale")
```

Arguments

`llocation, lscale`
: Character. Parameter link functions for location parameter a and scale parameter b. See `Links` for more choices.

`ilocation, iscale`
: Optional initial values. If given, it must be numeric and values are recycled to the appropriate length. The default is to choose the value internally.

`imethod`
: Initialization method. Either the value 1 or 2.

`zero`
: See `CommonVGAMffArguments` for information.

Details

The Laplace distribution is often known as the *double-exponential* distribution and, for modelling, has heavier tail than the normal distribution. The Laplace density function is

$$f(y) = \frac{1}{2b} \exp\left(-\frac{|y-a|}{b}\right)$$

where $-\infty < y < \infty$, $-\infty < a < \infty$ and $b > 0$. Its mean is a and its variance is $2b^2$. This parameterization is called the *classical Laplace distribution* by Kotz et al. (2001), and the density is symmetric about a.

For `y ~ 1` (where `y` is the response) the maximum likelihood estimate (MLE) for the location parameter is the sample median, and the MLE for b is `mean(abs(y-location))` (replace location by its MLE if unknown).

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Warning

This family function has not been fully tested. The MLE regularity conditions do *not* hold for this distribution, therefore misleading inferences may result, e.g., in the `summary` and `vcov` of the object. Hence this family function might be withdrawn from **VGAM** in the future.

Note

This family function uses Fisher scoring. Convergence may be slow for non-intercept-only models; half-stepping is frequently required.

Author(s)

T. W. Yee

References

Kotz, S., Kozubowski, T. J. and Podgorski, K. (2001). *The Laplace distribution and generalizations: a revisit with applications to communications, economics, engineering, and finance*, Boston: Birkhauser.

See Also

rlaplace, alaplace2 (which differs slightly from this parameterization), exponential, median.

Examples

```
ldata <- data.frame(y = rlaplace(nn <- 100, 2, scale = exp(1)))
fit <- vglm(y  ~ 1, laplace, ldata, trace = TRUE)
coef(fit, matrix = TRUE)
Coef(fit)
with(ldata, median(y))

ldata <- data.frame(x = runif(nn <- 1001))
ldata <- transform(ldata, y = rlaplace(nn, 2, scale = exp(-1 + 1*x)))
coef(vglm(y ~ x, laplace(iloc = 0.2, imethod = 2, zero = 1), ldata,
          trace = TRUE), matrix = TRUE)
```

---

laplaceUC *The Laplace Distribution*

Description

Density, distribution function, quantile function and random generation for the Laplace distribution with location parameter location and scale parameter scale.

Usage

```
dlaplace(x, location = 0, scale = 1, log = FALSE)
plaplace(q, location = 0, scale = 1, lower.tail = TRUE, log.p = FALSE)
qlaplace(p, location = 0, scale = 1, lower.tail = TRUE, log.p = FALSE)
rlaplace(n, location = 0, scale = 1)
```

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
n	number of observations. Same as in runif.
location	the location parameter a, which is the mean.
scale	the scale parameter b. Must consist of positive values.
log	Logical. If log = TRUE then the logarithm of the density is returned.
lower.tail, log.p	Same meaning as in pnorm or qnorm.

Details

The Laplace distribution is often known as the double-exponential distribution and, for modelling, has heavier tail than the normal distribution. The Laplace density function is

$$f(y) = \frac{1}{2b} \exp\left(-\frac{|y-a|}{b} \right)$$

where $-\infty < y < \infty$, $-\infty < a < \infty$ and $b > 0$. The mean is a and the variance is $2b^2$.

See `laplace`, the **VGAM** family function for estimating the two parameters by maximum likelihood estimation, for formulae and details. Apart from `n`, all the above arguments may be vectors and are recyled to the appropriate length if necessary.

Value

`dlaplace` gives the density, `plaplace` gives the distribution function, `qlaplace` gives the quantile function, and `rlaplace` generates random deviates.

Author(s)

T. W. Yee and Kai Huang

References

Forbes, C., Evans, M., Hastings, N. and Peacock, B. (2011). *Statistical Distributions*, Hoboken, NJ, USA: John Wiley and Sons, Fourth edition.

See Also

`laplace`.

Examples

```
loc <- 1; b <- 2
y <- rlaplace(n = 100, loc = loc, scale = b)
mean(y)  # sample mean
loc      # population mean
var(y)   # sample variance
2 * b^2  # population variance

## Not run:  loc <- 0; b <- 1.5; x <- seq(-5, 5, by = 0.01)
plot(x, dlaplace(x, loc, b), type = "l", col = "blue",
     main = "Blue is density, orange is the CDF", ylim = c(0,1),
     sub = "Purple are 5,10,...,95 percentiles", las = 1, ylab = "")
abline(h = 0, col = "blue", lty = 2)
lines(qlaplace(seq(0.05,0.95,by = 0.05), loc, b),
      dlaplace(qlaplace(seq(0.05, 0.95, by = 0.05), loc, b), loc, b),
      col = "purple", lty = 3, type = "h")
lines(x, plaplace(x, loc, b), type = "l", col = "orange")
abline(h = 0, lty = 2)
## End(Not run)

plaplace(qlaplace(seq(0.05, 0.95, by = 0.05), loc, b), loc, b)
```

`latvar` *Latent Variables*

Description

Generic function for the *latent variables* of a model.

Usage

```
latvar(object, ...)
    lv(object, ...)
```

Arguments

`object`	An object for which the extraction of latent variables is meaningful.
`...`	Other arguments fed into the specific methods function of the model. Sometimes they are fed into the methods function for `Coef`.

Details

Latent variables occur in reduced-rank regression models, as well as in quadratic and additive ordination models. For the latter two, latent variable values are often called *site scores* by ecologists. Latent variables are linear combinations of the explanatory variables.

Value

The value returned depends specifically on the methods function invoked.

Warning

`latvar` and `lv` are identical, but the latter will be deprecated soon.

Latent variables are not really applicable to `vglm`/`vgam` models.

Author(s)

Thomas W. Yee

References

Yee, T. W. and Hastie, T. J. (2003). Reduced-rank vector generalized linear models. *Statistical Modelling*, **3**, 15–41.

Yee, T. W. (2004). A new technique for maximum-likelihood canonical Gaussian ordination. *Ecological Monographs*, **74**, 685–701.

Yee, T. W. (2006). Constrained additive ordination. *Ecology*, **87**, 203–213.

See Also

`latvar.qrrvglm`, `latvar.rrvglm`, `latvar.cao`, `lvplot`.

Examples

```
## Not run:
hspider[, 1:6] <- scale(hspider[, 1:6])  # Standardized environmental vars
set.seed(123)
p1 <- cao(cbind(Pardlugu, Pardmont, Pardnigr, Pardpull, Zoraspin) ~
          WaterCon + BareSand + FallTwig + CoveMoss + CoveHerb + ReflLux,
          family = poissonff, data = hspider, Rank = 1, df1.nl =
          c(Zoraspin = 2.5, 3), Bestof = 3, Crow1positive = TRUE)

var(latvar(p1))  # Scaled to unit variance  # Scaled to unit variance
c(latvar(p1))    # Estimated site scores

## End(Not run)
```

---

leipnik *Leipnik Regression Family Function*

Description

Estimates the two parameters of a (transformed) Leipnik distribution by maximum likelihood estimation.

Usage

```
leipnik(lmu = "logitlink", llambda = logofflink(offset = 1),
        imu = NULL, ilambda = NULL)
```

Arguments

lmu, llambda	Link function for the μ and λ parameters. See Links for more choices.
imu, ilambda	Numeric. Optional initial values for μ and λ.

Details

The (transformed) Leipnik distribution has density function

$$f(y; \mu, \lambda) = \frac{\{y(1-y)\}^{-\frac{1}{2}}}{\mathrm{Beta}(\frac{\lambda+1}{2}, \frac{1}{2})} \left[1 + \frac{(y-\mu)^2}{y(1-y)}\right]^{-\frac{\lambda}{2}}$$

where $0 < y < 1$ and $\lambda > -1$. The mean is μ (returned as the fitted values) and the variance is $1/\lambda$.

Jorgensen (1997) calls the above the **transformed** Leipnik distribution, and if $y = (x + 1)/2$ and $\mu = (\theta + 1)/2$, then the distribution of X as a function of x and θ is known as the the (untransformed) Leipnik distribution. Here, both x and θ are in $(-1, 1)$.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, rrvglm and vgam.

Note

Convergence may be slow or fail. Until better initial value estimates are forthcoming try assigning the argument `ilambda` some numerical value if it fails to converge. Currently, Newton-Raphson is implemented, not Fisher scoring. Currently, this family function probably only really works for intercept-only models, i.e., `y ~ 1` in the formula.

Author(s)

T. W. Yee

References

Jorgensen, B. (1997). *The Theory of Dispersion Models*. London: Chapman & Hall

Johnson, N. L. and Kotz, S. and Balakrishnan, N. (1995). *Continuous Univariate Distributions*, 2nd edition, Volume 2, New York: Wiley. (pages 612–617).

See Also

`mccullagh89`.

Examples

```
ldata <- data.frame(y = rnorm(2000, 0.5, 0.1))  # Improper data
fit <- vglm(y ~ 1, leipnik(ilambda = 1), ldata, trace = TRUE)
head(fitted(fit))
with(ldata, mean(y))
summary(fit)
coef(fit, matrix = TRUE)
Coef(fit)

sum(weights(fit))  # Sum of the prior weights
sum(weights(fit, type = "work"))  # Sum of the working weights
```

---

lerch *Lerch Phi Function*

Description

Computes the Lerch Phi function.

Usage

```
lerch(x, s, v, tolerance = 1.0e-10, iter = 100)
```

Arguments

x, s, v	Numeric. This function recyles values of x, s, and v if necessary.
tolerance	Numeric. Accuracy required, must be positive and less than 0.01.
iter	Maximum number of iterations allowed to obtain convergence. If iter is too small then a result of NA may occur; if so, try increasing its value.

Details

Also known as the Lerch transcendent, it can be defined by an integral involving analytical continuation. An alternative definition is the series

$$\Phi(x, s, v) = \sum_{n=0}^{\infty} \frac{x^n}{(n+v)^s}$$

which converges for $|x| < 1$ as well as for $|x| = 1$ with $s > 1$. The series is undefined for integers $v <= 0$. Actually, x may be complex but this function only works for real x. The algorithm used is based on the relation

$$\Phi(x, s, v) = x^m \Phi(x, s, v + m) + \sum_{n=0}^{m-1} \frac{x^n}{(n+v)^s}.$$

See the URL below for more information. This function is a wrapper function for the C code described below.

Value

Returns the value of the function evaluated at the values of x, s, v. If the above ranges of x and v are not satisfied, or some numeric problems occur, then this function will return an NA for those values. (The C code returns 6 possible return codes, but this is not passed back up to the R level.)

Warning

This function has not been thoroughly tested and contains limitations, for example, the zeta function cannot be computed with this function even though $\zeta(s) = \Phi(x = 1, s, v = 1)$. Several numerical problems can arise, such as lack of convergence, overflow and underflow, especially near singularities. If any problems occur then an NA will be returned. For example, if $|x| = 1$ and $s > 1$ then convergence may be so slow that changing tolerance and/or iter may be needed to get an answer (that is treated cautiously).

Note

There are a number of special cases, e.g., the Riemann zeta-function is $\zeta(s) = \Phi(x = 1, s, v = 1)$. Another example is the Hurwitz zeta function $\zeta(s, v) = \Phi(x = 1, s, v = v)$. The special case of $s = 1$ corresponds to the hypergeometric 2F1, and this is implemented in the **gsl** package. The Lerch Phi function should not be confused with the Lerch zeta function though they are quite similar.

Author(s)

S. V. Aksenov and U. D. Jentschura wrote the C code (called Version 1.00). The R wrapper function was written by T. Yee.

References

Originally the code was found at `http://aksenov.freeshell.org/lerchphi/source/lerchphi.c`.

Bateman, H. (1953). *Higher Transcendental Functions*. Volume 1. McGraw-Hill, NY, USA.

See Also

`zeta`.

Examples

```
## Not run:
s <- 2; v <- 1; x <- seq(-1.1, 1.1, length = 201)
plot(x, lerch(x, s = s, v = v), type = "l", col = "blue",
     las = 1, main = paste0("lerch(x, s = ", s,", v = ", v, ")"))
abline(v = 0, h = 1, lty = "dashed", col = "gray")

## End(Not run)
```

`leukemia` *Acute Myelogenous Leukemia Survival Data*

Description

Survival in patients with Acute Myelogenous Leukemia

Usage

```
data(leukemia)
```

Format

time:	survival or censoring time
status:	censoring status
x:	maintenance chemotherapy given? (factor)

Note

This data set has been transferred from **survival** and renamed from `aml` to `leukemia`.

Source

Rupert G. Miller (1997). *Survival Analysis*. John Wiley & Sons.

`levy` *Levy Distribution Family Function*

Description

Estimates the scale parameter of the Levy distribution by maximum likelihood estimation.

Usage

```
levy(location = 0, lscale = "loglink", iscale = NULL)
```

Arguments

`location`	Location parameter. Must have a known value. Called a below.
`lscale`	Parameter link function for the (positive) scale parameter b. See `Links` for more choices.
`iscale`	Initial value for the b parameter. By default, an initial value is chosen internally.

Details

The Levy distribution is one of three stable distributions whose density function has a tractable form. The formula for the density is

$$f(y;b) = \sqrt{\frac{b}{2\pi}} \exp\left(\frac{-b}{2(y-a)}\right) / (y-a)^{3/2}$$

where $a < y < \infty$ and $b > 0$. Note that if a is very close to `min(y)` (where `y` is the response), then numerical problem will occur. The mean does not exist. The median is returned as the fitted values.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Author(s)

T. W. Yee

References

Nolan, J. P. (2005). *Stable Distributions: Models for Heavy Tailed Data*.

See Also

The Nolan article was at `http://academic2.american.edu/~jpnolan/stable/chap1.pdf`.

Examples

```
nn <- 1000; loc1 <- 0; loc2 <- 10
myscale <- 1  # log link ==> 0 is the answer
ldata <-
  data.frame(y1 = loc1 + myscale/rnorm(nn)^2,  # Levy(myscale, a)
             y2 = rlevy(nn, loc = loc2, scale = exp(+2)))
# Cf. Table 1.1 of Nolan for Levy(1,0)
with(ldata, sum(y1 > 1) / length(y1))  # Should be 0.6827
with(ldata, sum(y1 > 2) / length(y1))  # Should be 0.5205

fit1 <- vglm(y1 ~ 1, levy(location = loc1), ldata, trace = TRUE)
coef(fit1, matrix = TRUE)
Coef(fit1)
summary(fit1)
head(weights(fit1, type = "work"))

fit2 <- vglm(y2 ~ 1, levy(location = loc2), ldata, trace = TRUE)
coef(fit2, matrix = TRUE)
Coef(fit2)
c(median = with(ldata, median(y2)),
  fitted.median = head(fitted(fit2), 1))
```

`lgamma1` *Log-gamma Distribution Family Function*

Description

Estimation of the parameter of the standard and nonstandard log-gamma distribution.

Usage

```
lgamma1(lshape = "loglink", ishape = NULL)
lgamma3(llocation = "identitylink", lscale = "loglink",
   lshape = "loglink", ilocation = NULL, iscale = NULL, ishape = 1,
   zero = c("scale", "shape"))
```

Arguments

`llocation, lscale`
: Parameter link function applied to the location parameter a and the positive scale parameter b. See `Links` for more choices.

`lshape`
: Parameter link function applied to the positive shape parameter k. See `Links` for more choices.

`ishape`
: Initial value for k. If given, it must be positive. If failure to converge occurs, try some other value. The default means an initial value is determined internally.

`ilocation, iscale`
: Initial value for a and b. The defaults mean an initial value is determined internally for each.

zero — An integer-valued vector specifying which linear/additive predictors are modelled as intercepts only. The values must be from the set {1,2,3}. The default value means none are modelled as intercept-only terms. See CommonVGAMffArguments for more information.

Details

The probability density function of the standard log-gamma distribution is given by

$$f(y;k) = \exp[ky - \exp(y)]/\Gamma(k),$$

for parameter $k > 0$ and all real y. The mean of Y is `digamma(k)` (returned as the fitted values) and its variance is `trigamma(k)`.

For the non-standard log-gamma distribution, one replaces y by $(y - a)/b$, where a is the location parameter and b is the positive scale parameter. Then the density function is

$$f(y) = \exp[k(y - a)/b - \exp((y - a)/b)]/(b\,\Gamma(k)).$$

The mean and variance of Y are `a + b*digamma(k)` (returned as the fitted values) and `b^2 * trigamma(k)`, respectively.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Note

The standard log-gamma distribution can be viewed as a generalization of the standard type 1 extreme value density: when $k = 1$ the distribution of $-Y$ is the standard type 1 extreme value distribution.

The standard log-gamma distribution is fitted with `lgamma1` and the non-standard (3-parameter) log-gamma distribution is fitted with `lgamma3`.

Author(s)

T. W. Yee

References

Kotz, S. and Nadarajah, S. (2000). *Extreme Value Distributions: Theory and Applications*, pages 48–49, London: Imperial College Press.

Johnson, N. L. and Kotz, S. and Balakrishnan, N. (1995). *Continuous Univariate Distributions*, 2nd edition, Volume 2, p.89, New York: Wiley.

See Also

rlgamma, gengamma.stacy, prentice74, gamma1, lgamma.

Examples

```
ldata <- data.frame(y = rlgamma(100, shape = exp(1)))
fit <- vglm(y ~ 1, lgamma1, ldata, trace = TRUE, crit = "coef")
summary(fit)
coef(fit, matrix = TRUE)
Coef(fit)

ldata <- data.frame(x2 = runif(nn <- 5000))  # Another example
ldata <- transform(ldata, loc = -1 + 2 * x2, Scale = exp(1))
ldata <- transform(ldata, y = rlgamma(nn, loc, sc = Scale, sh = exp(0)))
fit2 <- vglm(y ~ x2, lgamma3, data = ldata, trace = TRUE, crit = "c")
coef(fit2, matrix = TRUE)
```

---

lgammaUC *The Log-Gamma Distribution*

Description

Density, distribution function, quantile function and random generation for the log-gamma distribution with location parameter `location`, scale parameter `scale` and shape parameter `k`.

Usage

```
dlgamma(x, location = 0, scale = 1, shape = 1, log = FALSE)
plgamma(q, location = 0, scale = 1, shape = 1,
        lower.tail = TRUE, log.p = FALSE)
qlgamma(p, location = 0, scale = 1, shape = 1,
        lower.tail = TRUE, log.p = FALSE)
rlgamma(n, location = 0, scale = 1, shape = 1)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. Same as `runif`.
`location`	the location parameter a.
`scale`	the (positive) scale parameter b.
`shape`	the (positive) shape parameter k.
`log`	Logical. If `log = TRUE` then the logarithm of the density is returned.
`lower.tail, log.p`	Same meaning as in `pnorm` or `qnorm`.

Details

See `lgamma1`, the **VGAM** family function for estimating the one parameter standard log-gamma distribution by maximum likelihood estimation, for formulae and other details. Apart from `n`, all the above arguments may be vectors and are recyled to the appropriate length if necessary.

Value

dlgamma gives the density, plgamma gives the distribution function, qlgamma gives the quantile function, and rlgamma generates random deviates.

Note

The **VGAM** family function lgamma3 is for the three parameter (nonstandard) log-gamma distribution.

Author(s)

T. W. Yee and Kai Huang

References

Kotz, S. and Nadarajah, S. (2000). *Extreme Value Distributions: Theory and Applications*, pages 48–49, London: Imperial College Press.

See Also

lgamma1, prentice74.

Examples

```
## Not run:  loc <- 1; Scale <- 1.5; shape <- 1.4
x <- seq(-3.2, 5, by = 0.01)
plot(x, dlgamma(x, loc = loc, Scale, shape = shape), type = "l",
     col = "blue", ylim = 0:1,
     main = "Blue is density, orange is the CDF",
     sub = "Red are 5,10,...,95 percentiles", las = 1, ylab = "")
abline(h = 0, col = "blue", lty = 2)
lines(qlgamma(seq(0.05, 0.95, by = 0.05), loc = loc, Scale, sh = shape),
      dlgamma(qlgamma(seq(0.05, 0.95, by = 0.05), loc = loc, sc = Scale,
                      shape = shape),
    loc = loc, Scale, shape = shape), col = "red", lty = 3, type = "h")
lines(x, plgamma(x, loc = loc, Scale, shape = shape), col = "orange")
abline(h = 0, lty = 2)
## End(Not run)
```

Lindley *The Lindley Distribution*

Description

Density, cumulative distribution function, and random generation for the Lindley distribution.

Usage

```
dlind(x, theta, log = FALSE)
plind(q, theta, lower.tail = TRUE, log.p = FALSE)
rlind(n, theta)
```

Arguments

`x, q`	vector of quantiles.
`n`	number of observations. Same as in `runif`.
`log`	Logical. If `log = TRUE` then the logarithm of the density is returned.
`theta`	positive parameter.
`lower.tail, log.p`	Same meaning as in `pnorm` or `qnorm`.

Details

See `lindley` for details.

Value

`dlind` gives the density, `plind` gives the cumulative distribution function, and `rlind` generates random deviates.

Author(s)

T. W. Yee and Kai Huang

See Also

`lindley`.

Examples

```
theta <- exp(-1); x <- seq(0.0, 17, length = 700)
dlind(0:10, theta)
## Not run:
plot(x, dlind(x, theta), type = "l", las = 1, col = "blue",
     main = "dlind(x, theta = exp(-1))")
abline(h = 1, col = "grey", lty = "dashed")
## End(Not run)
```

lindley	*1-parameter Gamma Distribution*

Description

Estimates the (1-parameter) Lindley distribution by maximum likelihood estimation.

Usage

```
lindley(link = "loglink", itheta = NULL, zero = NULL)
```

Arguments

`link`	Link function applied to the (positive) parameter. See `Links` for more choices.
`itheta, zero`	See `CommonVGAMffArguments` for information.

Details

The density function is given by

$$f(y; \theta) = \theta^2 (1 + y) \exp(-\theta y)/(1 + \theta)$$

for $\theta > 0$ and $y > 0$. The mean of Y (returned as the fitted values) is $\mu = (\theta + 2)/(\theta(\theta + 1))$. The variance is $(\theta^2 + 4\theta + 2)/(\theta(\theta + 1))^2$.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Note

This **VGAM** family function can handle multiple responses (inputted as a matrix). Fisher scoring is implemented.

Author(s)

T. W. Yee

References

Lindley, D. V. (1958). Fiducial distributions and Bayes' theorem. *Journal of the Royal Statistical Society, Series B, Methodological*, **20**, 102–107.

Ghitany, M. E. and Atieh, B. and Nadarajah, S. (2008). Lindley distribution and its application. *Math. Comput. Simul.*, **78**, 493–506.

See Also

dlind, gammaR, simulate.vlm.

Examples

```
ldata <- data.frame(y = rlind(n = 1000, theta = exp(3)))
fit <- vglm(y ~ 1, lindley, data = ldata, trace = TRUE, crit = "coef")
coef(fit, matrix = TRUE)
Coef(fit)
summary(fit)
```

---

linkfun *Link Functions for VGLMs*

Description

Returns the link functions, and parameter names, for *vector generalized linear models* (VGLMs).

Usage

```
linkfun(object, ...)
linkfunvlm(object, earg = FALSE, ...)
```

Arguments

object	An object which has parameter link functions, e.g., has class "vglm".
earg	Logical. Return the extra arguments associated with each link function? If TRUE then a list is returned.
...	Arguments that might be used in the future.

Details

All fitted VGLMs have a link function applied to each parameter. This function returns these, and optionally, the extra arguments associated with them.

Value

Usually just a (named) character string, with the link functions in order. It is named with the parameter names. If earg = TRUE then a list with the following components.

link	The default output.
earg	The extra arguments, in order.

Note

Presently, the multinomial logit model has only one link function, multilogitlink, so a warning is not issued for that link. For other models, if the number of link functions does not equal M then a warning may be issued.

Author(s)

Thomas W. Yee

See Also

linkfun, multilogitlink, vglm.

Examples

```
pneumo <- transform(pneumo, let = log(exposure.time))
fit1 <- vglm(cbind(normal, mild, severe) ~ let, propodds, data = pneumo)
coef(fit1, matrix = TRUE)
linkfun(fit1)
linkfun(fit1, earg = TRUE)

fit2 <- vglm(cbind(normal, mild, severe) ~ let, multinomial, data = pneumo)
coef(fit2, matrix = TRUE)
linkfun(fit2)
linkfun(fit2, earg = TRUE)
```

Links *Link functions for VGLM/VGAM/etc. families*

Description

The **VGAM** package provides a number of (parameter) link functions which are described in general here. Collectively, they offer the user considerable choice and flexibility for modelling data.

Usage

```
TypicalVGAMlink(theta, someParameter = 0, bvalue = NULL, inverse = FALSE,
                deriv = 0, short = TRUE, tag = FALSE)
```

Arguments

theta	Numeric or character. This is usually θ (default) but can sometimes be η, depending on the other arguments. If theta is character then inverse and deriv are ignored. The name theta should always be the name of the first argument.
someParameter	Some parameter, e.g., an offset.
bvalue	Boundary value, positive if given. If 0 < theta then values of theta which are less than or equal to 0 can be replaced by bvalue before computing the link function value. Values of theta which are greater than or equal to 1 can be replaced by 1 minus bvalue before computing the link function value. The value bvalue = .Machine$double.eps is sometimes a reasonable value, or something slightly higher.
inverse	Logical. If TRUE and deriv = 0 then the inverse link value θ is returned, hence the argument theta is really η. In all other cases, the argument theta is really θ.

`deriv` Integer. Either 0, 1, or 2, specifying the order of the derivative. Some link functions handle values up to 3 or 4.

`short, tag` Logical. These are used for labelling the `blurb` slot of a `vglmff-class` object. These arguments are used only if `theta` is character, and gives the formula for the link in character form. If `tag = TRUE` then the result is preceeded by a little more information.

Details

Almost all **VGAM** link functions have something similar to the argument list as given above. In this help file we have $\eta = g(\theta)$ where g is the link function, θ is the parameter and η is the linear/additive predictor. The link g must be strictly monotonic and twice-differentiable in its range.

The following is a brief enumeration of all **VGAM** link functions.

For parameters lying between 0 and 1 (e.g., probabilities): `logitlink`, `probitlink`, `clogloglink`, `cauchitlink`, `foldsqrtlink`, `logclink`.

For positive parameters (i.e., greater than 0): `loglink`, `negloglink`, `sqrtlink`, `powerlink`.

For parameters greater than 1: `loglogl ink`, `logloglogl ink` (greater than e).

For parameters between -1 and 1: `fisherzlink`, `rhobitlink`.

For parameters between finite A and B: `extlogitlink`, `logofflink` ($B = \infty$).

For unrestricted parameters (i.e., any value): `identitylink`, `negidentitylink`, `reciprocallink`, `negreciprocallink`.

Value

Returns one of: the link function value or its first or second derivative, the inverse link or its first or second derivative, or a character description of the link.

Here are the general details. If `inverse = FALSE` and `deriv = 0` (default) then the ordinary link function $\eta = g(\theta)$ is returned.

If `inverse = TRUE` and `deriv = 0` then the inverse link function value is returned, hence `theta` is really η (the only occasion this happens).

If `inverse = FALSE` and `deriv = 1` then it is $d\eta/d\theta$ *as a function of* θ. If `inverse = FALSE` and `deriv = 2` then it is $d^2\eta/d\theta^2$ *as a function of* θ.

If `inverse = TRUE` and `deriv = 1` then it is $d\theta/d\eta$ *as a function of* θ. If `inverse = TRUE` and `deriv = 2` then it is $d^2\theta/d\eta^2$ *as a function of* θ.

It is only when `deriv = 1` that `linkfun(theta, deriv = 1, inverse = TRUE)` and `linkfun(theta, deriv = 1, inverse = FALSE)` are *reciprocals* of each other. In particular, `linkfun(theta, deriv = 2, inverse = TRUE)` and `linkfun(theta, deriv = 2, inverse = FALSE)` are *not* reciprocals of each other in general.

Warning

The output of link functions changed at **VGAM** `0.9-9` (date was around 2015-07). Formerly, `linkfun(theta, deriv = 1)` is now `linkfun(theta, deriv = 1, inverse = TRUE)`, or equivalently, `1 / linkfun(theta, deriv = 1, inverse = TRUE)`. Also, formerly, `linkfun(theta, deriv = 2)` was 1 / linkfun(theta, deriv = 2, inverse = TRUE). This was a bug. Altogether, these are big changes and the user should beware!

In **VGAM** 1.0-7 (January 2019) all link function names were made to end in the characters "link", e.g., loglink replaces loge, logitlink replaces logit. For this most of them were renamed. Upward compatability holds for older link function names, however, users should adopt the new names immediately.

Note

VGAM link functions are generally not compatible with other functions outside the package. In particular, they won't work with glm or any other package for fitting GAMs.

From October 2006 onwards, all **VGAM** family functions will only contain one default value for each link argument rather than giving a vector of choices. For example, rather than binomialff(link = c("logitlink", "probitlink", "clogloglink", "cauchitlink", "identitylink"), ...) it is now binomialff(link = "logitlink", ...). No checking will be done to see if the user's choice is reasonable. This means that the user can write his/her own **VGAM** link function and use it within any **VGAM** family function. Altogether this provides greater flexibility. The downside is that the user must specify the *full* name of the link function, by either assigning the link argument the full name as a character string, or just the name itself. See the examples below.

From August 2012 onwards, a major change in link functions occurred. Argument esigma (and the like such as earg) used to be in **VGAM** prior to version 0.9-0 (released during the 2nd half of 2012). The major change is that arguments such as offset that used to be passed in via those arguments can done directly through the link function. For example, gev(lshape = "logofflink", eshape = list(offset = 0.5)) is replaced by gev(lshape = logofflink(offset = 0.5)). The @misc slot no longer has link and earg components, but two other components replace these. Functions such as dtheta.deta(), d2theta.deta2(), d3theta.deta3(), eta2theta(), theta2eta() are modified.

From January 2019 onwards, all link function names ended in "link". See above for details.

Author(s)

T. W. Yee

References

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*, 2nd ed. London: Chapman & Hall.

See Also

TypicalVGAMfamilyFunction, linkfun, vglm, vgam, rrvglm. cqo, cao.

Examples

```
logitlink("a")
logitlink("a", short = FALSE)
logitlink("a", short = FALSE, tag = TRUE)

logofflink(1:5, offset = 1)  # Same as log(1:5 + 1)
```

```
powerlink(1:5, power = 2)  # Same as (1:5)^2

## Not run:  # This is old and no longer works:
logofflink(1:5, earg = list(offset = 1))
powerlink(1:5, earg = list(power = 2))

## End(Not run)

fit1 <- vgam(agaaus ~ altitude,
             binomialff(link = "clogloglink"), hunua)  # best
fit2 <- vgam(agaaus ~ altitude,
             binomialff(link =  clogloglink ), hunua)  # okay

## Not run:
# This no longer works since "clog" is not a valid VGAM link function:
fit3 <- vgam(agaaus ~ altitude,
             binomialff(link = "clog"), hunua)  # not okay


# No matter what the link, the estimated var-cov matrix is the same
y <- rbeta(n = 1000, shape1 = exp(0), shape2 = exp(1))
fit1 <- vglm(y ~ 1, betaR(lshape1 = "identitylink",
                          lshape2 = "identitylink"),
             trace = TRUE, crit = "coef")
fit2 <- vglm(y ~ 1, betaR(lshape1 = logofflink(offset = 1.1),
                          lshape2 = logofflink(offset = 1.1)), trace=TRUE)
vcov(fit1, untransform = TRUE)
vcov(fit1, untransform = TRUE) -
vcov(fit2, untransform = TRUE)  # Should be all 0s
\dontrun{ # This is old:
fit1@misc$earg  # Some 'special' parameters
fit2@misc$earg  # Some 'special' parameters are here
}


par(mfrow = c(2, 2))
p <- seq(0.05, 0.95, len = 200)  # A rather restricted range
x <- seq(-4, 4, len = 200)
plot(p, logitlink(p), type = "l", col = "blue")
plot(x, logitlink(x, inverse = TRUE), type = "l", col = "blue")
plot(p, logitlink(p, deriv=1), type="l", col="blue") # 1 / (p*(1-p))
plot(p, logitlink(p, deriv=2), type="l", col="blue") # (2*p-1)/(p*(1-p))^2

## End(Not run)
```

`Lino` *The Generalized Beta Distribution (Libby and Novick, 1982)*

Description

Density, distribution function, quantile function and random generation for the generalized beta distribution, as proposed by Libby and Novick (1982).

Usage

```
dlino(x, shape1, shape2, lambda = 1, log = FALSE)
plino(q, shape1, shape2, lambda = 1, lower.tail = TRUE, log.p = FALSE)
qlino(p, shape1, shape2, lambda = 1, lower.tail = TRUE, log.p = FALSE)
rlino(n, shape1, shape2, lambda = 1)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. Same as in `runif`.
`shape1, shape2, lambda`	see `lino`.
`log`	Logical. If `log = TRUE` then the logarithm of the density is returned.
`lower.tail, log.p`	Same meaning as in `pnorm` or `qnorm`.

Details

See `lino`, the **VGAM** family function for estimating the parameters, for the formula of the probability density function and other details.

Value

`dlino` gives the density, `plino` gives the distribution function, `qlino` gives the quantile function, and `rlino` generates random deviates.

Author(s)

T. W. Yee and Kai Huang

See Also

`lino`.

Examples

```
## Not run:   lambda <- 0.4; shape1 <- exp(1.3); shape2 <- exp(1.3)
x <- seq(0.0, 1.0, len = 101)
plot(x, dlino(x, shape1 = shape1, shape2 = shape2, lambda = lambda),
     type = "l", col = "blue", las = 1, ylab = "",
     main = "Blue is PDF, orange is the CDF",
     sub = "Purple lines are the 10,20,...,90 percentiles")
abline(h = 0, col = "blue", lty = 2)
```

```
lines(x, plino(x, shape1, shape2, lambda = lambda), col = "orange")
probs <- seq(0.1, 0.9, by = 0.1)
Q <- qlino(probs, shape1 = shape1, shape2 = shape2, lambda = lambda)
lines(Q, dlino(Q, shape1 = shape1, shape2 = shape2, lambda = lambda),
      col = "purple", lty = 3, type = "h")
plino(Q, shape1, shape2, lambda = lambda) - probs  # Should be all 0

## End(Not run)
```

---

lino *Generalized Beta Distribution Family Function*

Description

Maximum likelihood estimation of the 3-parameter generalized beta distribution as proposed by Libby and Novick (1982).

Usage

```
lino(lshape1 = "loglink", lshape2 = "loglink", llambda = "loglink",
     ishape1 = NULL,   ishape2 = NULL,   ilambda = 1, zero = NULL)
```

Arguments

`lshape1, lshape2`
: Parameter link functions applied to the two (positive) shape parameters a and b. See `Links` for more choices.

`llambda`
: Parameter link function applied to the parameter λ. See `Links` for more choices.

`ishape1, ishape2, ilambda`
: Initial values for the parameters. A `NULL` value means one is computed internally. The argument `ilambda` must be numeric, and the default corresponds to a standard beta distribution.

`zero`
: Can be an integer-valued vector specifying which linear/additive predictors are modelled as intercepts only. Here, the values must be from the set {1,2,3} which correspond to a, b, λ, respectively. See `CommonVGAMffArguments` for more information.

Details

Proposed by Libby and Novick (1982), this distribution has density

$$f(y; a, b, \lambda) = \frac{\lambda^a y^{a-1} (1-y)^{b-1}}{B(a,b)\{1-(1-\lambda)y\}^{a+b}}$$

for $a > 0$, $b > 0$, $\lambda > 0$, $0 < y < 1$. Here B is the beta function (see `beta`). The mean is a complicated function involving the Gauss hypergeometric function. If X has a `lino` distribution with parameters `shape1`, `shape2`, `lambda`, then $Y = \lambda X/(1-(1-\lambda)X)$ has a standard beta distribution with parameters `shape1`, `shape2`.

Since $\log(\lambda) = 0$ corresponds to the standard beta distribution, a summary of the fitted model performs a t-test for whether the data belongs to a standard beta distribution (provided the loglink link for λ is used; this is the default).

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Note

The fitted values, which is usually the mean, have not been implemented yet. Currently the median is returned as the fitted values.

Although Fisher scoring is used, the working weight matrices are positive-definite only in a certain region of the parameter space. Problems with this indicate poor initial values or an ill-conditioned model or insufficient data etc.

This model is can be difficult to fit. A reasonably good value of ilambda seems to be needed so if the self-starting initial values fail, try experimenting with the initial value arguments. Experience suggests ilambda is better a little larger, rather than smaller, compared to the true value.

Author(s)

T. W. Yee

References

Libby, D. L. and Novick, M. R. (1982). Multivariate generalized beta distributions with applications to utility assessment. *Journal of Educational Statistics*, **7**, 271–294.

Gupta, A. K. and Nadarajah, S. (2004). *Handbook of Beta Distribution and Its Applications*, NY: Marcel Dekker, Inc.

See Also

Lino, genbetaII.

Examples

```
ldata <- data.frame(y1 = rbeta(n = 1000, exp(0.5), exp(1)))  # Std beta
fit <- vglm(y1 ~ 1, lino, data = ldata, trace = TRUE)
coef(fit, matrix = TRUE)
Coef(fit)
head(fitted(fit))
summary(fit)

# Nonstandard beta distribution
ldata <- transform(ldata, y2 = rlino(1000, shape1 = exp(1),
                                     shape2 = exp(2), lambda = exp(1)))
fit2 <- vglm(y2 ~ 1,
             lino(lshape1 = "identitylink", lshape2 = "identitylink",
                  ilamb = 10), data = ldata, trace = TRUE)
```

```
coef(fit2, matrix = TRUE)
```

---

lirat *Low-iron Rat Teratology Data*

Description

Low-iron rat teratology data.

Usage

```
data(lirat)
```

Format

A data frame with 58 observations on the following 4 variables.

- `N` Litter size.
- `R` Number of dead fetuses.
- `hb` Hemoglobin level.
- `grp` Group number. Group 1 is the untreated (low-iron) group, group 2 received injections on day 7 or day 10 only, group 3 received injections on days 0 and 7, and group 4 received injections weekly.

Details

The following description comes from Moore and Tsiatis (1991). The data comes from the experimental setup from Shepard et al. (1980), which is typical of studies of the effects of chemical agents or dietary regimens on fetal development in laboratory rats.

Female rats were put in iron-deficient diets and divided into 4 groups. One group of controls was given weekly injections of iron supplement to bring their iron intake to normal levels, while another group was given only placebo injections. Two other groups were given fewer iron-supplement injections than the controls. The rats were made pregnant, sacrificed 3 weeks later, and the total number of fetuses and the number of dead fetuses in each litter were counted.

For each litter the number of dead fetuses may be considered to be Binomial(N, p) where N is the litter size and p is the probability of a fetus dying. The parameter p is expected to vary from litter to litter, therefore the total variance of the proportions will be greater than that predicted by a binomial model, even when the covariates for hemoglobin level and experimental group are accounted for.

Source

Moore, D. F. and Tsiatis, A. (1991) Robust Estimation of the Variance in Moment Methods for Extra-binomial and Extra-Poisson Variation. *Biometrics*, **47**, 383–401.

References

Shepard, T. H., Mackler, B. and Finch, C. A. (1980). Reproductive studies in the iron-deficient rat. *Teratology*, **22**, 329–334.

Examples

```
## Not run:
# cf. Figure 3 of Moore and Tsiatis (1991)
plot(R / N ~ hb, data = lirat, pch = as.character(grp), col = grp,
     las = 1, xlab = "Hemoglobin level", ylab = "Proportion Dead")
## End(Not run)
```

---

lms.bcg *LMS Quantile Regression with a Box-Cox transformation to a Gamma Distribution*

Description

LMS quantile regression with the Box-Cox transformation to the gamma distribution.

Usage

```
lms.bcg(percentiles = c(25, 50, 75), zero = c("lambda", "sigma"),
   llambda = "identitylink", lmu = "identitylink", lsigma = "loglink",
   idf.mu = 4, idf.sigma = 2, ilambda = 1, isigma = NULL)
```

Arguments

`percentiles` A numerical vector containing values between 0 and 100, which are the quantiles. They will be returned as 'fitted values'.

`zero` See `lms.bcn`. See `CommonVGAMffArguments` for more information.

`llambda, lmu, lsigma`
See `lms.bcn`.

`idf.mu, idf.sigma`
See `lms.bcn`.

`ilambda, isigma`
See `lms.bcn`.

Details

Given a value of the covariate, this function applies a Box-Cox transformation to the response to best obtain a gamma distribution. The parameters chosen to do this are estimated by maximum likelihood or penalized maximum likelihood. Similar details can be found at `lms.bcn`.

Value

An object of class "vglmff" (see `vglmff-class`). The object is used by modelling functions such as `vglm`, `rrvglm` and `vgam`.

Warning

This **VGAM** family function comes with the same warnings as `lms.bcn`. Also, the expected value of the second derivative with respect to lambda may be incorrect (my calculations do not agree with the Lopatatzidis and Green manuscript.)

Note

Similar notes can be found at lms.bcn.

Author(s)

Thomas W. Yee

References

Lopatatzidis A. and Green, P. J. (unpublished manuscript). Semiparametric quantile regression using the gamma distribution.

Yee, T. W. (2004). Quantile regression via vector generalized additive models. *Statistics in Medicine*, **23**, 2295–2315.

See Also

lms.bcn, lms.yjn, qtplot.lmscreg, deplot.lmscreg, cdf.lmscreg, bmi.nz, amlexponential.

Examples

```
# This converges, but deplot(fit) and qtplot(fit) do not work
fit0 <- vglm(BMI ~ sm.bs(age, df = 4), lms.bcg, bmi.nz, trace = TRUE)
coef(fit0, matrix = TRUE)
## Not run:
par(mfrow = c(1, 1))
plotvgam(fit0, se = TRUE)  # Plot mu function (only)

## End(Not run)

# Use a trick: fit0 is used for initial values for fit1.
fit1 <- vgam(BMI ~ s(age, df = c(4, 2)), etastart = predict(fit0),
             lms.bcg(zero = 1), bmi.nz, trace = TRUE)

# Difficult to get a model that converges.  Here, we prematurely
# stop iterations because it fails near the solution.
fit2 <- vgam(BMI ~ s(age, df = c(4, 2)), maxit = 4,
             lms.bcg(zero = 1, ilam = 3), bmi.nz, trace = TRUE)
summary(fit1)
head(predict(fit1))
head(fitted(fit1))
head(bmi.nz)
# Person 1 is near the lower quartile of BMI amongst people his age
head(cdf(fit1))

## Not run:
# Quantile plot
par(bty = "l", mar=c(5, 4, 4, 3) + 0.1, xpd = TRUE)
qtplot(fit1, percentiles=c(5, 50, 90, 99), main = "Quantiles",
       xlim = c(15, 90), las = 1, ylab = "BMI", lwd = 2, lcol = 4)

# Density plot
```

```
ygrid <- seq(15, 43, len = 100)  # BMI ranges
par(mfrow = c(1, 1), lwd = 2)
(aa <- deplot(fit1, x0 = 20, y = ygrid, xlab = "BMI", col = "black",
  main = "PDFs at Age = 20 (black), 42 (red) and 55 (blue)"))
aa <- deplot(fit1, x0 = 42, y = ygrid, add=TRUE, llty=2, col="red")
aa <- deplot(fit1, x0 = 55, y = ygrid, add=TRUE, llty=4, col="blue",
             Attach = TRUE)
aa@post$deplot  # Contains density function values

## End(Not run)
```

---

lms.bcn *LMS Quantile Regression with a Box-Cox Transformation to Normality*

Description

LMS quantile regression with the Box-Cox transformation to normality.

Usage

```
lms.bcn(percentiles = c(25, 50, 75), zero = c("lambda", "sigma"),
   llambda = "identitylink", lmu = "identitylink",
   lsigma = "loglink", idf.mu = 4, idf.sigma = 2, ilambda = 1,
   isigma = NULL, tol0 = 0.001)
```

Arguments

`percentiles` A numerical vector containing values between 0 and 100, which are the quantiles. They will be returned as 'fitted values'.

`zero` Can be an integer-valued vector specifying which linear/additive predictors are modelled as intercepts only. The values must be from the set {1,2,3}. The default value usually increases the chance of successful convergence. Setting `zero = NULL` means they all are functions of the covariates. For more information see `CommonVGAMffArguments`.

`llambda, lmu, lsigma` Parameter link functions applied to the first, second and third linear/additive predictors. See `Links` for more choices, and `CommonVGAMffArguments`.

`idf.mu` Degrees of freedom for the cubic smoothing spline fit applied to get an initial estimate of mu. See `vsmooth.spline`.

`idf.sigma` Degrees of freedom for the cubic smoothing spline fit applied to get an initial estimate of sigma. See `vsmooth.spline`. This argument may be assigned `NULL` to get an initial value using some other algorithm.

`ilambda` Initial value for lambda. If necessary, it is recycled to be a vector of length n where n is the number of (independent) observations.

`isigma` Optional initial value for sigma. If necessary, it is recycled to be a vector of length n. The default value, `NULL`, means an initial value is computed in the `@initialize` slot of the family function.

`tol0` Small positive number, the tolerance for testing if lambda is equal to zero.

Details

Given a value of the covariate, this function applies a Box-Cox transformation to the response to best obtain normality. The parameters chosen to do this are estimated by maximum likelihood or penalized maximum likelihood.

In more detail, the basic idea behind this method is that, for a fixed value of x, a Box-Cox transformation of the response Y is applied to obtain standard normality. The 3 parameters (λ, μ, σ, which start with the letters "L-M-S" respectively, hence its name) are chosen to maximize a penalized log-likelihood (with `vgam`). Then the appropriate quantiles of the standard normal distribution are back-transformed onto the original scale to get the desired quantiles. The three parameters may vary as a smooth function of x.

The Box-Cox power transformation here of the Y, given x, is

$$Z = [(Y/\mu(x))^{\lambda(x)} - 1]/(\sigma(x)\,\lambda(x))$$

for $\lambda(x) \neq 0$. (The singularity at $\lambda(x) = 0$ is handled by a simple function involving a logarithm.) Then Z is assumed to have a standard normal distribution. The parameter $\sigma(x)$ must be positive, therefore **VGAM** chooses $\eta(x)^T = (\lambda(x), \mu(x), \log(\sigma(x)))$ by default. The parameter μ is also positive, but while $\log(\mu)$ is available, it is not the default because μ is more directly interpretable. Given the estimated linear/additive predictors, the 100α percentile can be estimated by inverting the Box-Cox power transformation at the 100α percentile of the standard normal distribution.

Of the three functions, it is often a good idea to allow $\mu(x)$ to be more flexible because the functions $\lambda(x)$ and $\sigma(x)$ usually vary more smoothly with x. This is somewhat reflected in the default value for the argument `zero`, viz. `zero = c(1, 3)`.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, `rrvglm` and `vgam`.

Warning

The computations are not simple, therefore convergence may fail. Set `trace = TRUE` to monitor convergence if it isn't set already. Convergence failure will occur if, e.g., the response is bimodal at any particular value of x. In case of convergence failure, try different starting values. Also, the estimate may diverge quickly near the solution, in which case try prematurely stopping the iterations by assigning `maxits` to be the iteration number corresponding to the highest likelihood value.

One trick is to fit a simple model and use it to provide initial values for a more complex model; see in the examples below.

Note

The response must be positive because the Box-Cox transformation cannot handle negative values. In theory, the LMS-Yeo-Johnson-normal method can handle both positive and negative values.

In general, the lambda and sigma functions should be more smoother than the mean function. Having `zero = 1`, `zero = 3` or `zero = c(1, 3)` is often a good idea. See the example below.

Author(s)

Thomas W. Yee

References

Cole, T. J. and Green, P. J. (1992). Smoothing Reference Centile Curves: The LMS Method and Penalized Likelihood. *Statistics in Medicine*, **11**, 1305–1319.

Green, P. J. and Silverman, B. W. (1994). *Nonparametric Regression and Generalized Linear Models: A Roughness Penalty Approach*, London: Chapman & Hall.

Yee, T. W. (2004). Quantile regression via vector generalized additive models. *Statistics in Medicine*, **23**, 2295–2315.

See Also

`lms.bcg`, `lms.yjn`, `qtplot.lmscreg`, `deplot.lmscreg`, `cdf.lmscreg`, `eCDF`, `extlogF1`, `alaplace1`, `amlnormal`, `denorm`, `CommonVGAMffArguments`.

Examples

```
## Not run:  require("VGAMdata")
mysub <- subset(xs.nz, sex == "M" & ethnicity == "Maori" & study1)
mysub <- transform(mysub, BMI = weight / height^2)
BMIdata <- na.omit(mysub)
BMIdata <- subset(BMIdata, BMI < 80 & age < 65,
                   select = c(age, BMI))  # Delete an outlier
summary(BMIdata)

fit <- vgam(BMI ~ s(age, df = c(4, 2)), lms.bcn(zero = 1), BMIdata)

par(mfrow = c(1, 2))
plot(fit, scol = "blue", se = TRUE)  # The two centered smooths

head(predict(fit))
head(fitted(fit))
head(BMIdata)
head(cdf(fit))  # Person 46 is probably overweight, given his age
100 * colMeans(c(depvar(fit)) < fitted(fit))  # Empirical proportions

# Correct for "vgam" objects but not very elegant:
fit@family@linkinv(eta = predict(fit, data.frame(age = 60)),
   extra = list(percentiles = c(10, 50)))

if (FALSE) {
```

```
# These work for "vglm" objects:
fit2 <- vglm(BMI ~ bs(age, df = 4), lms.bcn(zero = 3), BMIdata)
predict(fit2, percentiles = c(10, 50),
        newdata = data.frame(age = 60), type = "response")
head(fitted(fit2, percentiles = c(10, 50)))  # Different percentiles
}

# Convergence problems? Use fit0 for initial values for fit1
fit0 <- vgam(BMI ~ s(age, df = 4), lms.bcn(zero = c(1, 3)), BMIdata)
fit1 <- vgam(BMI ~ s(age, df = c(4, 2)), lms.bcn(zero = 1), BMIdata,
            etastart = predict(fit0))

## End(Not run)

## Not run: # Quantile plot
par(bty = "l", mar = c(5, 4, 4, 3) + 0.1, xpd = TRUE)
qtplot(fit, percentiles = c(5, 50, 90, 99), main = "Quantiles",
       xlim = c(15, 66), las = 1, ylab = "BMI", lwd = 2, lcol = 4)

# Density plot
ygrid <- seq(15, 43, len = 100)  # BMI ranges
par(mfrow = c(1, 1), lwd = 2)
(aa <- deplot(fit, x0 = 20, y = ygrid, xlab = "BMI", col = "black",
  main = "PDFs at Age = 20 (black), 42 (red) and 55 (blue)"))
aa <- deplot(fit, x0 = 42, y = ygrid, add = TRUE, llty = 2, col = "red")
aa <- deplot(fit, x0 = 55, y = ygrid, add = TRUE, llty = 4, col = "blue",
             Attach = TRUE)
aa@post$deplot  # Contains density function values

## End(Not run)
```

lms.yjn — *LMS Quantile Regression with a Yeo-Johnson Transformation to Normality*

Description

LMS quantile regression with the Yeo-Johnson transformation to normality. This family function is experimental and the LMS-BCN family function is recommended instead.

Usage

```
lms.yjn(percentiles = c(25, 50, 75), zero = c("lambda", "sigma"),
   llambda = "identitylink", lsigma = "loglink",
   idf.mu = 4, idf.sigma = 2,
   ilambda = 1, isigma = NULL, rule = c(10, 5),
   yoffset = NULL, diagW = FALSE, iters.diagW = 6)
lms.yjn2(percentiles = c(25, 50, 75), zero = c("lambda", "sigma"),
   llambda = "identitylink", lmu = "identitylink", lsigma = "loglink",
   idf.mu = 4, idf.sigma = 2, ilambda = 1.0,
   isigma = NULL, yoffset = NULL, nsimEIM = 250)
```

Arguments

percentiles
: A numerical vector containing values between 0 and 100, which are the quantiles. They will be returned as 'fitted values'.

zero
: See lms.bcn. See CommonVGAMffArguments for more information.

llambda, lmu, lsigma
: See lms.bcn.

idf.mu, idf.sigma
: See lms.bcn.

ilambda, isigma
: See lms.bcn.

rule
: Number of abscissae used in the Gaussian integration scheme to work out elements of the weight matrices. The values given are the possible choices, with the first value being the default. The larger the value, the more accurate the approximation is likely to be but involving more computational expense.

yoffset
: A value to be added to the response y, for the purpose of centering the response before fitting the model to the data. The default value, NULL, means -median(y) is used, so that the response actually used has median zero. The yoffset is saved on the object and used during prediction.

diagW
: Logical. This argument is offered because the expected information matrix may not be positive-definite. Using the diagonal elements of this matrix results in a higher chance of it being positive-definite, however convergence will be very slow.

  If TRUE, then the first iters.diagW iterations will use the diagonal of the expected information matrix. The default is FALSE, meaning faster convergence.

iters.diagW
: Integer. Number of iterations in which the diagonal elements of the expected information matrix are used. Only used if diagW = TRUE.

nsimEIM
: See CommonVGAMffArguments for more information.

Details

Given a value of the covariate, this function applies a Yeo-Johnson transformation to the response to best obtain normality. The parameters chosen to do this are estimated by maximum likelihood or penalized maximum likelihood. The function lms.yjn2() estimates the expected information matrices using simulation (and is consequently slower) while lms.yjn() uses numerical integration. Try the other if one function fails.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm and vgam.

Warning

The computations are not simple, therefore convergence may fail. In that case, try different starting values.

The generic function predict, when applied to a lms.yjn fit, does not add back the yoffset value.

As described above, this family function is experimental and the LMS-BCN family function is recommended instead.

Note

The response may contain both positive and negative values. In contrast, the LMS-Box-Cox-normal and LMS-Box-Cox-gamma methods only handle a positive response because the Box-Cox transformation cannot handle negative values.

Some other notes can be found at `lms.bcn`.

Author(s)

Thomas W. Yee

References

Yeo, I.-K. and Johnson, R. A. (2000). A new family of power transformations to improve normality or symmetry. *Biometrika*, **87**, 954–959.

Yee, T. W. (2004). Quantile regression via vector generalized additive models. *Statistics in Medicine*, **23**, 2295–2315.

Yee, T. W. (2002). An Implementation for Regression Quantile Estimation. Pages 3–14. In: Haerdle, W. and Ronz, B., *Proceedings in Computational Statistics COMPSTAT 2002*. Heidelberg: Physica-Verlag.

See Also

`lms.bcn`, `lms.bcg`, `qtplot.lmscreg`, `deplot.lmscreg`, `cdf.lmscreg`, `bmi.nz`, `amlnormal`.

Examples

```
fit <- vgam(BMI ~ s(age, df = 4), lms.yjn, bmi.nz, trace = TRUE)
head(predict(fit))
head(fitted(fit))
head(bmi.nz)
# Person 1 is near the lower quartile of BMI amongst people his age
head(cdf(fit))

## Not run:
# Quantile plot
par(bty = "l", mar = c(5, 4, 4, 3) + 0.1, xpd = TRUE)
qtplot(fit, percentiles = c(5, 50, 90, 99), main = "Quantiles",
       xlim = c(15, 90), las = 1, ylab = "BMI", lwd = 2, lcol = 4)

# Density plot
ygrid <- seq(15, 43, len = 100)  # BMI ranges
par(mfrow = c(1, 1), lwd = 2)
(Z <- deplot(fit, x0 = 20, y = ygrid, xlab = "BMI", col = "black",
    main = "PDFs at Age = 20 (black), 42 (red) and 55 (blue)"))
Z <- deplot(fit, x0 = 42, y = ygrid, add = TRUE, llty = 2, col = "red")
Z <- deplot(fit, x0 = 55, y = ygrid, add = TRUE, llty = 4, col = "blue",
```

```
            Attach = TRUE)
with(Z@post, deplot)  # Contains PDF values; == a@post$deplot

## End(Not run)
```

Log *Logarithmic Distribution*

Description

Density, distribution function, quantile function, and random generation for the logarithmic distribution.

Usage

```
dlog(x, shape, log = FALSE)
plog(q, shape, lower.tail = TRUE, log.p = FALSE)
qlog(p, shape)
rlog(n, shape)
```

Arguments

`x, q, p, n, lower.tail`
: Same interpretation as in `runif`.

`shape`
: The shape parameter value c described in in `logff`.

`log, log.p`
: Logical. If `log.p = TRUE` then all probabilities `p` are given as `log(p)`.

Details

The details are given in `logff`.

Value

`dlog` gives the density, `plog` gives the distribution function, `qlog` gives the quantile function, and `rlog` generates random deviates.

Note

Given some response data, the **VGAM** family function `logff` estimates the parameter `shape`. For `plog()`, if argument `q` contains large values and/or `q` is long in length then the memory requirements may be very high. Very large values in `q` are handled by an approximation by Owen (1965).

Author(s)

T. W. Yee

References

Forbes, C., Evans, M., Hastings, N. and Peacock, B. (2011). *Statistical Distributions*, Hoboken, NJ, USA: John Wiley and Sons, Fourth edition.

See Also

`logff`, `Gaitdlog`, `Oilog`. `Otlog`.

Examples

```
dlog(1:20, 0.5)
rlog(20, 0.5)

## Not run:  shape <- 0.8; x <- 1:10
plot(x, dlog(x, shape = shape), type = "h", ylim = 0:1,
     sub = "shape=0.8", las = 1, col = "blue", ylab = "shape",
     main = "Logarithmic distribution: blue=PDF; orange=CDF")
lines(x + 0.1, plog(x, shape), col = "orange", lty = 3, type = "h")
## End(Not run)
```

`log1mexp` *Logarithms with an Unit Offset and Exponential Term*

Description

Computes `log(1 + exp(x))` and `log(1 - exp(-x))` accurately.

Usage

```
log1mexp(x)
log1pexp(x)
```

Arguments

x	A vector of reals (numeric). Complex numbers not allowed since `expm1` and `log1p` do not handle these.

Details

Computes `log(1 + exp(x))` and `log(1 - exp(-x))` accurately. An adjustment is made when x is away from 0 in value.

Value

`log1mexp(x)` gives the value of $\log(1 - \exp(-x))$.

`log1pexp(x)` gives the value of $\log(1 + \exp(x))$.

Note

If NA or NaN is present in the input, the corresponding output will be NA.

Author(s)

This is a direct translation of the function in Martin Maechler's (2012) paper by Xiangjie Xue and T. W. Yee.

References

Maechler, Martin (2012). Accurately Computing log(1-exp(-|a|)). Assessed from the **Rmpfr** package.

See Also

log1p, expm1, exp, log

Examples

```
x <-  c(10, 50, 100, 200, 400, 500, 800, 1000, 1e4, 1e5, 1e20, Inf, NA)
log1pexp(x)
log(1 + exp(x))  # Naive; suffers from overflow
log1mexp(x)
log(1 - exp(-x))
y <- -x
log1pexp(y)
log(1 + exp(y))  # Naive; suffers from inaccuracy
```

---

logclink *Complementary-log Link Function*

Description

Computes the Complementary-log Transformation, Including its Inverse and the First Two Derivatives.

Usage

```
logclink(theta, bvalue = NULL, inverse = FALSE, deriv = 0,
         short = TRUE, tag = FALSE)
```

Arguments

theta	Numeric or character. See below for further details.
bvalue	See Links.
inverse, deriv, short, tag	Details at Links.

Details

The complementary-log link function is suitable for parameters that are less than unity. Numerical values of `theta` close to 1 or out of range result in `Inf`, `-Inf`, `NA` or `NaN`.

Value

For `deriv = 0`, the log of `theta`, i.e., `log(1-theta)` when `inverse = FALSE`, and if `inverse = TRUE` then `1-exp(theta)`.

For `deriv = 1`, then the function returns *d* `eta` / *d* `theta` as a function of `theta` if `inverse = FALSE`, else if `inverse = TRUE` then it returns the reciprocal.

Here, all logarithms are natural logarithms, i.e., to base *e*.

Note

Numerical instability may occur when `theta` is close to 1. One way of overcoming this is to use `bvalue`.

Author(s)

Thomas W. Yee

References

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*, 2nd ed. London: Chapman & Hall.

See Also

`Links`, `loglink`, `clogloglink`, `logloglink`, `logofflink`.

Examples

```
## Not run:
logclink(seq(-0.2, 1.1, by = 0.1))  # Has NAs

## End(Not run)
logclink(seq(-0.2,1.1,by=0.1),bvalue=1-.Machine$double.eps) # Has no NAs
```

`logF` *Natural Exponential Family Generalized Hyperbolic Secant Distribution Family Function*

Description

Maximum likelihood estimation of the 2-parameter log F distribution.

Usage

```
logF(lshape1 = "loglink", lshape2 = "loglink",
     ishape1 = NULL, ishape2 = 1, imethod = 1)
```

Arguments

`lshape1, lshape2`
: Parameter link functions for the shape parameters. Called α and β respectively. See `Links` for more choices.

`ishape1, ishape2`
: Optional initial values for the shape parameters. If given, it must be numeric and values are recycled to the appropriate length. The default is to choose the value internally. See `CommonVGAMffArguments` for more information.

`imethod`
: Initialization method. Either the value 1, 2, or .... See `CommonVGAMffArguments` for more information.

Details

The density for this distribution is

$$f(y; \alpha, \beta) = \exp(\alpha y)/[B(\alpha, \beta)(1 + e^y)^{\alpha+\beta}]$$

where y is real, $\alpha > 0$, $\beta > 0$, $B(., .)$ is the beta function `beta`.

Value

An object of class "vglmff" (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Author(s)

Thomas W. Yee

References

Jones, M. C. (2008). On a class of distributions with simple exponential tails. *Statistica Sinica*, **18**(3), 1101–1110.

See Also

`dlogF`, `extlogF1`, `logff`.

Examples

```
nn <- 1000
ldata <- data.frame(y1 = rnorm(nn, +1, sd = exp(2)),  # Not proper data
                    x2 = rnorm(nn, -1, sd = exp(2)),
                    y2 = rnorm(nn, -1, sd = exp(2)))  # Not proper data
fit1 <- vglm(y1 ~ 1 , logF, ldata, trace = TRUE)
fit2 <- vglm(y2 ~ x2, logF, ldata, trace = TRUE)
```

```
coef(fit2, matrix = TRUE)
summary(fit2)
vcov(fit2)

head(fitted(fit1))
with(ldata, mean(y1))
max(abs(head(fitted(fit1)) - with(ldata, mean(y1))))
```

---

logff *Logarithmic Distribution*

Description

Estimating the (single) parameter of the logarithmic distribution.

Usage

```
logff(lshape = "logitlink", gshape = -expm1(-7 * ppoints(4)),
      zero = NULL)
```

Arguments

lshape	Parameter link function for the parameter c, which lies between 0 and 1. See Links for more choices and information. Soon logfflink() will hopefully be available for event-rate data.
gshape, zero	Details at CommonVGAMffArguments. Practical experience shows that having the initial value for c being close to the solution is quite important.

Details

The logarithmic distribution is a generalized power series distribution that is based specifically on the logarithmic series (scaled to a probability function). Its probability function is $f(y) = ac^y/y$, for $y = 1, 2, 3, \ldots$, where $0 < c < 1$ (called shape), and $a = -1/\log(1-c)$. The mean is $ac/(1-c)$ (returned as the fitted values) and variance is $ac(1-ac)/(1-c)^2$. When the sample mean is large, the value of c tends to be very close to 1, hence it could be argued that logitlink is not the best choice.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Note

The function `log` computes the natural logarithm. In the **VGAM** library, a link function with option `loglink` corresponds to this.

Multiple responses are permitted.

The "logarithmic distribution" has various meanings in the literature. Sometimes it is also called the *log-series distribution*. Some others call some continuous distribution on $[a, b]$ by the name "logarithmic distribution".

Author(s)

T. W. Yee

References

Johnson N. L., Kemp, A. W. and Kotz S. (2005). *Univariate Discrete Distributions*, 3rd edition, ch.7. Hoboken, New Jersey: Wiley.

Forbes, C., Evans, M., Hastings, N. and Peacock, B. (2011) *Statistical Distributions*, Hoboken, NJ, USA: John Wiley and Sons, Fourth edition.

See Also

`Log`, `gaitdlog`, `oalog`, `oilog`, `otlog`, `log`, `loglink`, `logofflink`, `explogff`, `simulate.vlm`.

Examples

```
nn <- 1000
ldata <- data.frame(y = rlog(nn, shape = logitlink(0.2, inv = TRUE)))
fit <- vglm(y ~ 1, logff, data = ldata, trace = TRUE, crit = "c")
coef(fit, matrix = TRUE)
Coef(fit)
## Not run: with(ldata, spikeplot(y, col = "blue", capped = TRUE))
x <- seq(1, with(ldata, max(y)), by = 1)
with(ldata, lines(x + 0.1, dlog(x, Coef(fit)[1]), col = "orange",
        type = "h", lwd = 2))
## End(Not run)

# Example: Corbet (1943) butterfly Malaya data
corbet <- data.frame(nindiv = 1:24,
                 ofreq = c(118, 74, 44, 24, 29, 22, 20, 19, 20, 15, 12,
                           14, 6, 12, 6, 9, 9, 6, 10, 10, 11, 5, 3, 3))
fit <- vglm(nindiv ~ 1, logff, data = corbet, weights = ofreq)
coef(fit, matrix = TRUE)
shapehat <- Coef(fit)["shape"]
pdf2 <- dlog(x = with(corbet, nindiv), shape = shapehat)
print(with(corbet, cbind(nindiv, ofreq, fitted = pdf2 * sum(ofreq))),
      digits = 1)
```

`logistic` *Logistic Distribution Family Function*

Description

Estimates the location and scale parameters of the logistic distribution by maximum likelihood estimation.

Usage

```
logistic1(llocation = "identitylink", scale.arg = 1, imethod = 1)
logistic(llocation = "identitylink", lscale = "loglink",
         ilocation = NULL, iscale = NULL, imethod = 1, zero = "scale")
```

Arguments

`llocation, lscale`
: Parameter link functions applied to the location parameter l and scale parameter s. See `Links` for more choices, and `CommonVGAMffArguments` for more information.

`scale.arg`
: Known positive scale parameter (called s below).

`ilocation, iscale`
: See `CommonVGAMffArguments` for information.

`imethod, zero`
: See `CommonVGAMffArguments` for information.

Details

The two-parameter logistic distribution has a density that can be written as

$$f(y; l, s) = \frac{\exp[-(y-l)/s]}{s\left(1 + \exp[-(y-l)/s]\right)^2}$$

where $s > 0$ is the scale parameter, and l is the location parameter. The response $-\infty < y < \infty$. The mean of Y (which is the fitted value) is l and its variance is $\pi^2 s^2/3$.

A logistic distribution with `scale = 0.65` (see `dlogis`) resembles `dt` with `df = 7`; see `logistic1` and `studentt`.

`logistic1` estimates the location parameter only while `logistic` estimates both parameters. By default, $\eta_1 = l$ and $\eta_2 = \log(s)$ for `logistic`.

`logistic` can handle multiple responses.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, `rrvglm` and `vgam`.

Note

Fisher scoring is used, and the Fisher information matrix is diagonal.

Author(s)

T. W. Yee

References

Johnson, N. L. and Kotz, S. and Balakrishnan, N. (1994). *Continuous Univariate Distributions*, 2nd edition, Volume 1, New York: Wiley. Chapter 15.

Forbes, C., Evans, M., Hastings, N. and Peacock, B. (2011). *Statistical Distributions*, Hoboken, NJ, USA: John Wiley and Sons, Fourth edition.

Castillo, E., Hadi, A. S., Balakrishnan, N. and Sarabia, J. S. (2005). *Extreme Value and Related Models with Applications in Engineering and Science*, Hoboken, NJ, USA: Wiley-Interscience, p.130.

deCani, J. S. and Stine, R. A. (1986). A Note on Deriving the Information Matrix for a Logistic Distribution, *The American Statistician*, **40**, 220–222.

See Also

rlogis, CommonVGAMffArguments, logitlink, gensh, cumulative, bilogistic, simulate.vlm.

Examples

```
# Location unknown, scale known
ldata <- data.frame(x2 = runif(nn <- 500))
ldata <- transform(ldata, y1 = rlogis(nn, loc = 1+5*x2, sc = exp(2)))
fit1 <- vglm(y1 ~ x2, logistic1(scale = exp(2)), ldata, trace = TRUE)
coef(fit1, matrix = TRUE)

# Both location and scale unknown
ldata <- transform(ldata, y2 = rlogis(nn, loc = 1 + 5*x2, exp(x2)))
fit2 <- vglm(cbind(y1, y2) ~ x2, logistic, data = ldata, trace = TRUE)
coef(fit2, matrix = TRUE)
vcov(fit2)
summary(fit2)
```

---

logitlink *Logit Link Function*

Description

Computes the logit transformation, including its inverse and the first two derivatives.

Usage

```
logitlink(theta, bvalue = NULL, inverse = FALSE, deriv = 0,
    short = TRUE, tag = FALSE)
extlogitlink(theta, min = 0, max = 1, bminvalue = NULL,
    bmaxvalue = NULL, inverse = FALSE, deriv = 0,
    short = TRUE, tag = FALSE)
```

Arguments

`theta` Numeric or character. See below for further details.

`bvalue, bminvalue, bmaxvalue` See `Links`. These are boundary values. For `extlogitlink`, values of `theta` less than or equal to A or greater than or equal to B can be replaced by `bminvalue` and `bmaxvalue`.

`min, max` For `extlogitlink`, `min` gives A, `max` gives B, and for out of range values, `bminvalue` and `bmaxvalue`.

`inverse, deriv, short, tag` Details at `Links`.

Details

The logit link function is very commonly used for parameters that lie in the unit interval. It is the inverse CDF of the logistic distribution. Numerical values of `theta` close to 0 or 1 or out of range result in `Inf`, `-Inf`, `NA` or `NaN`.

The *extended* logit link function `extlogitlink` should be used more generally for parameters that lie in the interval (A, B), say. The formula is

$$\log((\theta - A)/(B - \theta))$$

and the default values for A and B correspond to the ordinary logit function. Numerical values of `theta` close to A or B or out of range result in `Inf`, `-Inf`, `NA` or `NaN`. However these can be replaced by values $bminvalue$ and $bmaxvalue$ first before computing the link function.

Value

For `logitlink` with `deriv = 0`, the logit of `theta`, i.e., `log(theta/(1-theta))` when `inverse = FALSE`, and if `inverse = TRUE` then `exp(theta)/(1+exp(theta))`.

For `deriv = 1`, then the function returns *d* `eta` / *d* `theta` as a function of `theta` if `inverse = FALSE`, else if `inverse = TRUE` then it returns the reciprocal.

Here, all logarithms are natural logarithms, i.e., to base *e*.

Note

Numerical instability may occur when `theta` is close to 1 or 0 (for `logitlink`), or close to A or B for `extlogitlink`. One way of overcoming this is to use, e.g., `bvalue`.

In terms of the threshold approach with cumulative probabilities for an ordinal response this link function corresponds to the univariate logistic distribution (see `logistic`).

Author(s)

Thomas W. Yee

References

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*, 2nd ed. London: Chapman & Hall.

See Also

Links, alogitlink, asinlink, logitoffsetlink, probitlink, clogloglink, cauchitlink, logistic1, loglink, Logistic, multilogitlink.

Examples

```
p <- seq(0.01, 0.99, by = 0.01)
logitlink(p)
max(abs(logitlink(logitlink(p), inverse = TRUE) - p))  # 0?

p <- c(seq(-0.02, 0.02, by = 0.01), seq(0.97, 1.02, by = 0.01))
logitlink(p)  # Has NAs
logitlink(p, bvalue = .Machine$double.eps)  # Has no NAs

p <- seq(0.9, 2.2, by = 0.1)
extlogitlink(p, min = 1, max = 2,
             bminvalue = 1 + .Machine$double.eps,
             bmaxvalue = 2 - .Machine$double.eps)  # Has no NAs

## Not run:  par(mfrow = c(2,2), lwd = (mylwd <- 2))
y <- seq(-4, 4, length = 100)
p <- seq(0.01, 0.99, by = 0.01)
for (d in 0:1) {
  myinv <- (d > 0)
  matplot(p, cbind( logitlink(p, deriv = d, inv = myinv),
                   probitlink(p, deriv = d, inv = myinv)), las = 1,
          type = "n", col = "purple", ylab = "transformation",
          main = if (d ==  0) "Some probability link functions"
          else "1 / first derivative")
  lines(p,   logitlink(p, deriv = d, inverse = myinv), col = "limegreen")
  lines(p,  probitlink(p, deriv = d, inverse = myinv), col = "purple")
  lines(p, clogloglink(p, deriv = d, inverse = myinv), col = "chocolate")
  lines(p, cauchitlink(p, deriv = d, inverse = myinv), col = "tan")
  if (d ==  0) {
    abline(v = 0.5, h = 0, lty = "dashed")
    legend(0, 4.5, c("logitlink", "probitlink",
           "clogloglink", "cauchitlink"), col = c("limegreen", "purple",
           "chocolate", "tan"), lwd = mylwd)
  } else
    abline(v = 0.5, lty = "dashed")
}

for (d in 0) {
```

```
  matplot(y, cbind(logitlink(y, deriv = d, inverse = TRUE),
                   probitlink(y, deriv = d, inverse = TRUE)), las = 1,
          type = "n", col = "purple", xlab = "transformation", ylab = "p",
          main = if (d ==  0) "Some inverse probability link functions"
          else "First derivative")
  lines(y,   logitlink(y, deriv = d, inv = TRUE), col = "limegreen")
  lines(y,  probitlink(y, deriv = d, inv = TRUE), col = "purple")
  lines(y, clogloglink(y, deriv = d, inv = TRUE), col = "chocolate")
  lines(y, cauchitlink(y, deriv = d, inv = TRUE), col = "tan")
  if (d ==  0) {
    abline(h = 0.5, v = 0, lty = "dashed")
    legend(-4, 1, c("logitlink", "probitlink", "clogloglink",
           "cauchitlink"), col = c("limegreen", "purple",
           "chocolate", "tan"), lwd = mylwd)
  }
}

p <- seq(0.21, 0.59, by = 0.01)
plot(p, extlogitlink(p, min = 0.2, max = 0.6), xlim = c(0, 1),
     type = "l", col = "black", ylab = "transformation",
     las = 1, main = "extlogitlink(p, min = 0.2, max = 0.6)")
par(lwd = 1)

## End(Not run)
```

`logitoffsetlink` *Logit-with-an-Offset Link Function*

Description

Computes the logitoffsetlink transformation, including its inverse and the first two derivatives.

Usage

```
logitoffsetlink(theta, offset = 0, inverse = FALSE, deriv = 0,
                 short = TRUE, tag = FALSE)
```

Arguments

`theta`	Numeric or character. See below for further details.
`offset`	The offset value(s), which must be non-negative. It is called K below.
`inverse, deriv, short, tag`	Details at `Links`.

Details

This link function allows for some asymmetry compared to the ordinary `logitlink` link. The formula is

$$\log(\theta/(1-\theta) - K)$$

and the default value for the offset K is corresponds to the ordinary `logitlink` link. When `inverse = TRUE` will mean that the value will lie in the interval $(K/(1 + K), 1)$.

Value

For `logitoffsetlink` with `deriv = 0`, the logitoffsetlink of `theta`, i.e., `log(theta/(1-theta) - K)` when `inverse = FALSE`, and if `inverse = TRUE` then `(K + exp(theta))/(1 + exp(theta) + K)`.

For `deriv = 1`, then the function returns *d* `eta` / *d* `theta` as a function of `theta` if `inverse = FALSE`, else if `inverse = TRUE` then it returns the reciprocal.

Here, all logarithms are natural logarithms, i.e., to base *e*.

Note

This function is numerical less stability than `logitlink`.

Author(s)

Thomas W. Yee

References

Komori, O. and Eguchi, S. et al., 2016. An asymmetric logistic model for ecological data. *Methods in Ecology and Evolution*, **7**.

See Also

`Links`, `logitlink`.

Examples

```
p <- seq(0.05, 0.99, by = 0.01); myoff <- 0.05
logitoffsetlink(p, myoff)
max(abs(logitoffsetlink(logitoffsetlink(p, myoff),
        myoff, inverse = TRUE) - p))  # Should be 0
```

`loglaplace` *Log-Laplace and Logit-Laplace Distribution Family Functions*

Description

Maximum likelihood estimation of the 1-parameter log-Laplace and the 1-parameter logit-Laplace distributions. These may be used for quantile regression for counts and proportions respectively.

Usage

```
loglaplace1(tau = NULL, llocation = "loglink",
    ilocation = NULL, kappa = sqrt(tau/(1 - tau)), Scale.arg = 1,
    ishrinkage = 0.95, parallel.locat = FALSE, digt = 4,
    idf.mu = 3, rep0 = 0.5, minquantile = 0, maxquantile = Inf,
    imethod = 1, zero = NULL)
logitlaplace1(tau = NULL, llocation = "logitlink",
    ilocation = NULL, kappa = sqrt(tau/(1 - tau)),
    Scale.arg = 1, ishrinkage = 0.95, parallel.locat = FALSE,
    digt = 4, idf.mu = 3, rep01 = 0.5, imethod = 1, zero = NULL)
```

Arguments

`tau, kappa` See `alaplace1`.

`llocation` Character. Parameter link functions for location parameter ξ. See `Links` for more choices. However, this argument should be left unchanged with count data because it restricts the quantiles to be positive. With proportions data `llocation` can be assigned a link such as `logitlink`, `probitlink`, `clogloglink`, etc.

`ilocation` Optional initial values. If given, it must be numeric and values are recycled to the appropriate length. The default is to choose the value internally.

`parallel.locat` Logical. Should the quantiles be parallel on the transformed scale (argument `llocation`)? Assigning this argument to `TRUE` circumvents the seriously embarrassing quantile crossing problem.

`imethod` Initialization method. Either the value 1, 2, or ....

`idf.mu, ishrinkage, Scale.arg, digt, zero`
See `alaplace1`. See `CommonVGAMffArguments` for information.

`rep0, rep01` Numeric, positive. Replacement values for 0s and 1s respectively. For count data, values of the response whose value is 0 are replaced by `rep0`; it avoids computing `log(0)`. For proportions data values of the response whose value is 0 or 1 are replaced by `min(rangey01[1]/2, rep01/w[y<=0])` and `max((1 + rangey01[2])/2, 1-rep01/w[y >= 1])` respectively; e.g., it avoids computing `logitlink(0)` or `logitlink(1)`. Here, `rangey01` is the 2-vector `range(y[(y > 0) & (y < 1)])` of the response.

`minquantile, maxquantile`
Numeric. The minimum and maximum values possible in the quantiles. These argument are effectively ignored by default since `loglink` keeps all quantiles positive. However, if `llocation = logofflink(offset = 1)` then it is possible that the fitted quantiles have value 0 because `minquantile = 0`.

Details

These **VGAM** family functions implement translations of the asymmetric Laplace distribution (ALD). The resulting variants may be suitable for quantile regression for count data or sample proportions. For example, a log link applied to count data is assumed to follow an ALD. Another example is a logit link applied to proportions data so as to follow an ALD. A positive random variable Y is said to have a log-Laplace distribution if $Y = e^W$ where W has an ALD. There are many variants of ALDs and the one used here is described in `alaplace1`.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm and vgam.

In the extra slot of the fitted object are some list components which are useful. For example, the sample proportion of values which are less than the fitted quantile curves, which is sum(wprior[y <= location]) / sum(wprior) internally. Here, wprior are the prior weights (called ssize below), y is the response and location is a fitted quantile curve. This definition comes about naturally from the transformed ALD data.

Warning

The **VGAM** family function logitlaplace1 will not handle a vector of just 0s and 1s as the response; it will only work satisfactorily if the number of trials is large.

See alaplace1 for other warnings. Care is needed with tau values which are too small, e.g., for count data the sample proportion of zeros must be less than all values in tau. Similarly, this also holds with logitlaplace1, which also requires all tau values to be less than the sample proportion of ones.

Note

The form of input for logitlaplace1 as response is a vector of proportions (values in $[0, 1]$) and the number of trials is entered into the weights argument of vglm/vgam. See Example 2 below. See alaplace1 for other notes in general.

Author(s)

Thomas W. Yee

References

Kotz, S., Kozubowski, T. J. and Podgorski, K. (2001). *The Laplace distribution and generalizations: a revisit with applications to communications, economics, engineering, and finance*, Boston: Birkhauser.

Kozubowski, T. J. and Podgorski, K. (2003). Log-Laplace distributions. *International Mathematical Journal*, **3**, 467–495.

Yee, T. W. (2020). Quantile regression for counts and proportions. In preparation.

See Also

alaplace1, dloglap.

Examples

```
# Example 1: quantile regression of counts with regression splines
set.seed(123); my.k <- exp(0)
adata <- data.frame(x2 = sort(runif(n <- 500)))
mymu <- function(x) exp( 1 + 3*sin(2*x) / (x+0.5)^2)
adata <- transform(adata, y = rnbinom(n, mu = mymu(x2), my.k))
```

```
mytau <- c(0.1, 0.25, 0.5, 0.75, 0.9); mydof = 3
# halfstepping is usual:
fitp <- vglm(y ~ sm.bs(x2, df = mydof), data = adata, trace = TRUE,
            loglaplace1(tau = mytau, parallel.locat = TRUE))

## Not run:  par(las = 1)  # Plot on a log1p() scale
mylwd <- 1.5

plot(jitter(log1p(y), factor = 1.5) ~ x2, adata, col = "red",
     pch = "o", cex = 0.75,
     main = "Example 1; green=truth, blue=estimated")
with(adata, matlines(x2, log1p(fitted(fitp)), col = "blue",
                     lty = 1, lwd = mylwd))
finexgrid <- seq(0, 1, len = 201)
for (ii in 1:length(mytau))
  lines(finexgrid, col = "green", lwd = mylwd,
        log1p(qnbinom(mytau[ii], mu = mymu(finexgrid), my.k)))

## End(Not run)
fitp@extra  # Contains useful information


# Example 2: sample proportions
set.seed(123); nnn <- 1000; ssize <- 100  # ssize = 1 wont work!
adata <- data.frame(x2 = sort(runif(nnn)))
mymu <- function(x) logitlink( 1.0 + 4*x, inv = TRUE)
adata <- transform(adata, ssize = ssize,
                   y2 = rbinom(nnn, ssize, prob = mymu(x2)) / ssize)

mytau <- c(0.25, 0.50, 0.75)
fit1 <- vglm(y2 ~ sm.bs(x2, df = 3),
        logitlaplace1(tau = mytau, lloc = "cloglogLink", paral = TRUE),
        data = adata, weights = ssize, trace = TRUE)

## Not run:
# Check the solution.  Note: this is like comparing apples with oranges.
plotvgam(fit1, se = TRUE, scol = "red", lcol = "blue",
         main = "Truth = 'green'")
# Centered approximately !
linkFunctionChar <- as.character(fit1@misc$link)
adata <- transform(adata, trueFunction =
           theta2eta(theta = mymu(x2), link = linkFunctionChar))
with(adata, lines(x2, trueFunction - mean(trueFunction), col = "green"))

# Plot the data + fitted quantiles (on the original scale)
myylim <- with(adata, range(y2))
plot(y2 ~ x2, adata, col = "blue", ylim = myylim, las = 1,
     pch = ".", cex = 2.5)
with(adata, matplot(x2, fitted(fit1), add = TRUE, lwd = 3, type = "l"))
truecol <- rep(1:3, len = fit1@misc$M)  # Add the 'truth'
smallxgrid <- seq(0, 1, len = 501)
for (ii in 1:length(mytau))
  lines(smallxgrid, col = truecol[ii], lwd = 2,
```

```
        qbinom(mytau[ii], pr = mymu(smallxgrid), si = ssize) / ssize)

# Plot on the eta (== logitlink()/probitlink()/...) scale
  with(adata, matplot(x2, predict(fit1), lwd = 3, type = "l"))
# Add the 'truth'
for (ii in 1:length(mytau)) {
  true.quant <- qbinom(mytau[ii], prob = mymu(smallxgrid),
                       size = ssize) / ssize
  lines(smallxgrid, theta2eta(true.quant, link = linkFunctionChar),
        col = truecol[ii], lwd = 2)
}
## End(Not run)
```

---

loglapUC *The Log-Laplace Distribution*

Description

Density, distribution function, quantile function and random generation for the 3-parameter log-Laplace distribution with location parameter `location.ald`, scale parameter `scale.ald` (on the log scale), and asymmetry parameter `kappa`.

Usage

```
dloglap(x, location.ald = 0, scale.ald = 1,
        tau = 0.5, kappa = sqrt(tau/(1-tau)), log = FALSE)
ploglap(q, location.ald = 0, scale.ald = 1, tau = 0.5,
        kappa = sqrt(tau/(1-tau)), lower.tail = TRUE, log.p = FALSE)
qloglap(p, location.ald = 0, scale.ald = 1, tau = 0.5,
        kappa = sqrt(tau/(1-tau)), lower.tail = TRUE, log.p = FALSE)
rloglap(n, location.ald = 0, scale.ald = 1,
        tau = 0.5, kappa = sqrt(tau/(1-tau)))
```

Arguments

- `x, q` — vector of quantiles.
- `p` — vector of probabilities.
- `n` — number of observations. If `length(n) > 1` then the length is taken to be the number required.
- `location.ald, scale.ald` — the location parameter ξ and the (positive) scale parameter σ, on the log scale.
- `tau` — the quantile parameter τ. Must consist of values in $(0, 1)$. This argument is used to specify `kappa` and is ignored if `kappa` is assigned.
- `kappa` — the asymmetry parameter κ. Must consist of positive values.
- `log` — if `TRUE`, probabilities `p` are given as `log(p)`.
- `lower.tail, log.p` — Same meaning as in `pnorm` or `qnorm`.

Details

A positive random variable Y is said to have a log-Laplace distribution if $\log(Y)$ has an asymmetric Laplace distribution (ALD). There are many variants of ALDs and the one used here is described in alaplace3.

Value

dloglap gives the density, ploglap gives the distribution function, qloglap gives the quantile function, and rloglap generates random deviates.

Author(s)

T. W. Yee and Kai Huang

References

Kozubowski, T. J. and Podgorski, K. (2003). Log-Laplace distributions. *International Mathematical Journal*, **3**, 467–495.

See Also

dalap, alaplace3, loglaplace1.

Examples

```
loc <- 0; sigma <- exp(0.5); kappa <- 1
x <- seq(-0.2, 5, by = 0.01)
## Not run: plot(x, dloglap(x, loc, sigma, kappa = kappa),
     type = "l", col = "blue", ylim = c(0,1),
     main = "Blue is density, red is the CDF",
     sub = "Purple are 5,10,...,95 percentiles", las = 1, ylab = "")
abline(h = 0, col = "blue", lty = 2)
lines(qloglap(seq(0.05,0.95,by = 0.05), loc, sigma, kappa = kappa),
  dloglap(qloglap(seq(0.05,0.95,by = 0.05), loc, sigma, kappa = kappa),
              loc, sigma, kappa = kappa),
      col = "purple", lty = 3, type = "h")
lines(x, ploglap(x, loc, sigma, kappa = kappa), type = "l", col = 2)
abline(h = 0, lty = 2)

## End(Not run)
ploglap(qloglap(seq(0.05,0.95,by = 0.05), loc, sigma, kappa = kappa),
        loc, sigma, kappa = kappa)
```

logLik.vlm	*Extract Log-likelihood for VGLMs/VGAMs/etc.*

Description

Calculates the log-likelihood value or the element-by-element contributions of the log-likelihood.

Usage

```
## S3 method for class 'vlm'
logLik(object, summation = TRUE, ...)
```

Arguments

`object`	Some **VGAM** object, for example, having class `vglmff-class`.
`summation`	Logical, apply `sum`? If `FALSE` then a n-vector or n-row matrix (with the number of responses as the number of columns) is returned. Each element is the contribution to the log-likelihood.
`...`	Currently unused. In the future: other possible arguments fed into `logLik` in order to compute the log-likelihood.

Details

By default, this function returns the log-likelihood of the object. Thus this code relies on the log-likelihood being defined, and computed, for the object.

Value

Returns the log-likelihood of the object. If `summation = FALSE` then a n-vector or n-row matrix (with the number of responses as the number of columns) is returned. Each element is the contribution to the log-likelihood. The prior weights are assimulated within the answer.

Warning

Not all **VGAM** family functions have had the `summation` checked.

Note

Not all **VGAM** family functions currently have the `summation` argument implemented.

Author(s)

T. W. Yee.

See Also

VGLMs are described in `vglm-class`; VGAMs are described in `vgam-class`; RR-VGLMs are described in `rrvglm-class`; `AIC`; `anova.vglm`.

Examples

```
zdata <- data.frame(x2 = runif(nn <- 50))
zdata <- transform(zdata, Ps01    = logitlink(-0.5       , inverse = TRUE),
                          Ps02    = logitlink( 0.5       , inverse = TRUE),
                          lambda1 =  loglink(-0.5 + 2*x2, inverse = TRUE),
                          lambda2 =  loglink( 0.5 + 2*x2, inverse = TRUE))
zdata <- transform(zdata, y1 = rzipois(nn, lambda = lambda1, pstr0 = Ps01),
                          y2 = rzipois(nn, lambda = lambda2, pstr0 = Ps02))

with(zdata, table(y1))  # Eyeball the data
with(zdata, table(y2))
fit2 <- vglm(cbind(y1, y2) ~ x2, zipoisson(zero = NULL), data = zdata)

logLik(fit2)  # Summed over the two responses
sum(logLik(fit2, sum = FALSE))  # For checking purposes
(ll.matrix <- logLik(fit2, sum = FALSE))  # nn x 2 matrix
colSums(ll.matrix)  # log-likelihood for each response
```

`loglinb2` *Loglinear Model for Two Binary Responses*

Description

Fits a loglinear model to two binary responses.

Usage

```
loglinb2(exchangeable = FALSE, zero = "u12")
```

Arguments

`exchangeable`	Logical. If `TRUE`, the two marginal probabilities are constrained to be equal. Should be set `TRUE` for ears, eyes, etc. data.
`zero`	Which linear/additive predictors are modelled as intercept-only? A `NULL` means none of them. See `CommonVGAMffArguments` for more information.

Details

The model is

$$P(Y_1 = y_1, Y_2 = y_2) = \exp(u_0 + u_1 y_1 + u_2 y_2 + u_{12} y_1 y_2)$$

where y_1 and y_2 are 0 or 1, and the parameters are u_1, u_2, u_{12}. The normalizing parameter u_0 can be expressed as a function of the other parameters, viz.,

$$u_0 = -\log[1 + \exp(u_1) + \exp(u_2) + \exp(u_1 + u_2 + u_{12})].$$

The linear/additive predictors are $(\eta_1, \eta_2, \eta_3)^T = (u_1, u_2, u_{12})^T$.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, rrvglm and vgam.

When fitted, the fitted.values slot of the object contains the four joint probabilities, labelled as (Y_1, Y_2) = (0,0), (0,1), (1,0), (1,1), respectively.

Note

The response must be a two-column matrix of ones and zeros only. This is more restrictive than binom2.or, which can handle more types of input formats. Note that each of the 4 combinations of the multivariate response need to appear in the data set. After estimation, the response attached to the object is also a two-column matrix; possibly in the future it might change into a four-column matrix.

Author(s)

Thomas W. Yee

References

Yee, T. W. and Wild, C. J. (2001). Discussion to: "Smoothing spline ANOVA for multivariate Bernoulli observations, with application to ophthalmology data (with discussion)" by Gao, F., Wahba, G., Klein, R., Klein, B. *Journal of the American Statistical Association*, **96**, 127–160.

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*, 2nd ed. London: Chapman & Hall.

See Also

binom2.or, binom2.rho, loglinb3.

Examples

```
coalminers <- transform(coalminers, Age = (age - 42) / 5)
# Get the n x 4 matrix of counts
fit0 <- vglm(cbind(nBnW,nBW,BnW,BW) ~ Age, binom2.or, coalminers)
counts <- round(c(weights(fit0, type = "prior")) * depvar(fit0))
# Create a n x 2 matrix response for loglinb2()
# bwmat <- matrix(c(0,0, 0,1, 1,0, 1,1), 4, 2, byrow = TRUE)
bwmat <- cbind(bln = c(0,0,1,1), wheeze = c(0,1,0,1))
matof1 <- matrix(1, nrow(counts), 1)
newminers <-
  data.frame(bln    = kronecker(matof1, bwmat[, 1]),
             wheeze = kronecker(matof1, bwmat[, 2]),
             wt     = c(t(counts)),
             Age    = with(coalminers, rep(age, rep(4, length(age)))))
newminers <- newminers[with(newminers, wt) > 0,]

fit <- vglm(cbind(bln,wheeze) ~ Age, loglinb2(zero = NULL),
            weight = wt, data = newminers)
coef(fit, matrix = TRUE)  # Same! (at least for the log odds-ratio)
```

```
summary(fit)

# Try reconcile this with McCullagh and Nelder (1989), p.234
(0.166-0.131) / 0.027458   # 1.275 is approximately 1.25
```

---

loglinb3 *Loglinear Model for Three Binary Responses*

Description

Fits a loglinear model to three binary responses.

Usage

```
loglinb3(exchangeable = FALSE, zero = c("u12", "u13", "u23"))
```

Arguments

`exchangeable`	Logical. If `TRUE`, the three marginal probabilities are constrained to be equal.
`zero`	Which linear/additive predictors are modelled as intercept-only? A `NULL` means none. See `CommonVGAMffArguments` for further information.

Details

The model is $P(Y_1 = y_1, Y_2 = y_2, Y_3 = y_3) =$

$$\exp(u_0 + u_1 y_1 + u_2 y_2 + u_3 y_3 + u_{12} y_1 y_2 + u_{13} y_1 y_3 + u_{23} y_2 y_3)$$

where y_1, y_2 and y_3 are 0 or 1, and the parameters are u_1, u_2, u_3, u_{12}, u_{13}, u_{23}. The normalizing parameter u_0 can be expressed as a function of the other parameters. Note that a third-order association parameter, u_{123} for the product $y_1 y_2 y_3$, is assumed to be zero for this family function.

The linear/additive predictors are $(\eta_1, \eta_2, \ldots, \eta_6)^T = (u_1, u_2, u_3, u_{12}, u_{13}, u_{23})^T$.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, `rrvglm` and `vgam`.

When fitted, the `fitted.values` slot of the object contains the eight joint probabilities, labelled as (Y_1, Y_2, Y_3) = (0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1), (1,1,0), (1,1,1), respectively.

Note

The response must be a 3-column matrix of ones and zeros only. Note that each of the 8 combinations of the multivariate response need to appear in the data set, therefore data sets will need to be large in order for this family function to work. After estimation, the response attached to the object is also a 3-column matrix; possibly in the future it might change into a 8-column matrix.

Author(s)

Thomas W. Yee

References

Yee, T. W. and Wild, C. J. (2001). Discussion to: "Smoothing spline ANOVA for multivariate Bernoulli observations, with application to ophthalmology data (with discussion)" by Gao, F., Wahba, G., Klein, R., Klein, B. *Journal of the American Statistical Association*, **96**, 127–160.

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*, 2nd ed. London: Chapman & Hall.

See Also

`loglinb2`, `binom2.or`, `hunua`.

Examples

```
lfit <- vglm(cbind(cyadea, beitaw, kniexc) ~ altitude, loglinb3,
             data = hunua, trace = TRUE)
coef(lfit, matrix = TRUE)
head(fitted(lfit))
summary(lfit)
```

`loglink` *Log Link Function, and Variants*

Description

Computes the log transformation, including its inverse and the first two derivatives.

Usage

```
loglink(theta, bvalue = NULL, inverse = FALSE, deriv = 0,
        short = TRUE, tag = FALSE)
negloglink(theta, bvalue = NULL, inverse = FALSE, deriv = 0,
           short = TRUE, tag = FALSE)
logneglink(theta, bvalue = NULL, inverse = FALSE, deriv = 0,
           short = TRUE, tag = FALSE)
```

Arguments

`theta` Numeric or character. See below for further details.

`bvalue` See `Links`.

`inverse, deriv, short, tag`
Details at `Links`.

Details

The log link function is very commonly used for parameters that are positive. Here, all logarithms are natural logarithms, i.e., to base e. Numerical values of `theta` close to 0 or out of range result in `Inf`, `-Inf`, `NA` or `NaN`.

The function `loglink` computes $\log(\theta)$ whereas `negloglink` computes $-\log(\theta) = \log(1/\theta)$.

The function `logneglink` computes $\log(-\theta)$, hence is suitable for parameters that are negative, e.g., a trap-shy effect in `posbernoulli.b`.

Value

The following concerns `loglink`. For `deriv = 0`, the log of `theta`, i.e., `log(theta)` when `inverse = FALSE`, and if `inverse = TRUE` then `exp(theta)`. For `deriv = 1`, then the function returns *d* `eta` / *d* `theta` as a function of `theta` if `inverse = FALSE`, else if `inverse = TRUE` then it returns the reciprocal.

Note

This function was called `loge` to avoid conflict with the `log` function. Numerical instability may occur when `theta` is close to 0 unless `bvalue` is used.

Author(s)

Thomas W. Yee

References

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*, 2nd ed. London: Chapman & Hall.

See Also

`Links`, `explink`, `logitlink`, `logclink`, `logloglink`, `log`, `logofflink`, `lambertW`, `posbernoulli.b`.

Examples

```
## Not run:  loglink(seq(-0.2, 0.5, by = 0.1))
 loglink(seq(-0.2, 0.5, by = 0.1), bvalue = .Machine$double.xmin)
negloglink(seq(-0.2, 0.5, by = 0.1))
negloglink(seq(-0.2, 0.5, by = 0.1), bvalue = .Machine$double.xmin)
## End(Not run)
logneglink(seq(-0.5, -0.2, by = 0.1))
```

logloglink *Log-log and Log-log-log Link Functions*

Description

Computes the two transformations, including their inverse and the first two derivatives.

Usage

```
logloglink(theta, bvalue = NULL, inverse = FALSE, deriv = 0,
           short = TRUE, tag = FALSE)
loglogloglink(theta, bvalue = NULL, inverse = FALSE, deriv = 0,
              short = TRUE, tag = FALSE)
```

Arguments

`theta`	Numeric or character. See below for further details.
`bvalue`	Values of `theta` which are less than or equal to 1 or e can be replaced by `bvalue` before computing the link function value. The component name `bvalue` stands for "boundary value". See `Links` for more information.
`inverse, deriv, short, tag`	Details at `Links`.

Details

The log-log link function is commonly used for parameters that are greater than unity. Similarly, the log-log-log link function is applicable for parameters that are greater than e. Numerical values of `theta` close to 1 or e or out of range result in `Inf`, `-Inf`, `NA` or `NaN`. One possible application of `loglogloglink()` is to the k parameter (also called `size`) of `negbinomial` to Poisson-like data but with only a small amount of overdispersion; then k is a large number relative to `munb`. In such situations a `loglink` or `loglog` link may not be sufficient to draw the estimate toward the interior of the parameter space. Using a more stronger link function can help mitigate the Hauck-Donner effect `hdeff`.

Value

For `logloglink()`: for `deriv = 0`, the log of `log(theta)`, i.e., `log(log(theta))` when `inverse = FALSE`, and if `inverse = TRUE` then `exp(exp(theta))`.

For `loglogloglink()`: for `deriv = 0`, the log of `log(log(theta))`, i.e., `log(log(log(theta)))` when `inverse = FALSE`, and if `inverse = TRUE` then `exp(exp(exp(theta)))`.

For `deriv = 1`, then the function returns *d* `theta` / *d* `eta` as a function of `theta` if `inverse = FALSE`, else if `inverse = TRUE` then it returns the reciprocal.

Here, all logarithms are natural logarithms, i.e., to base *e*.

Note

Numerical instability may occur when `theta` is close to 1 or e unless `bvalue` is used.

References

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*, 2nd ed. London: Chapman & Hall.

See Also

Links, loglink, logofflink.

Examples

```
x <- seq(0.8, 1.5, by = 0.1)
logloglink(x)  # Has NAs
logloglink(x, bvalue = 1.0 + .Machine$double.eps)  # Has no NAs

x <- seq(1.01, 10, len = 100)
logloglink(x)
max(abs(logloglink(logloglink(x), inverse = TRUE) - x))  # 0?
```

`lognormal` *Lognormal Distribution*

Description

Maximum likelihood estimation of the (univariate) lognormal distribution.

Usage

```
lognormal(lmeanlog = "identitylink", lsdlog = "loglink", zero = "sdlog")
```

Arguments

`lmeanlog, lsdlog`
: Parameter link functions applied to the mean and (positive) σ (standard deviation) parameter. Both of these are on the log scale. See Links for more choices.

`zero`
: Specifies which linear/additive predictor is modelled as intercept-only. For `lognormal()`, the values can be from the set {1,2} which correspond to `mu`, `sigma`, respectively. See CommonVGAMffArguments for more information.

Details

A random variable Y has a 2-parameter lognormal distribution if $\log(Y)$ is distributed $N(\mu, \sigma^2)$. The expected value of Y, which is

$$E(Y) = \exp(\mu + 0.5\sigma^2)$$

and not μ, make up the fitted values. The variance of Y is

$$Var(Y) = [\exp(\sigma^2) - 1] \exp(2\mu + \sigma^2).$$

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Author(s)

T. W. Yee

References

Kleiber, C. and Kotz, S. (2003). *Statistical Size Distributions in Economics and Actuarial Sciences*, Hoboken, NJ, USA: Wiley-Interscience.

See Also

Lognormal, uninormal, CommonVGAMffArguments, simulate.vlm.

Examples

```
ldata2 <- data.frame(x2 = runif(nn <- 1000))
ldata2 <- transform(ldata2, y1 = rlnorm(nn, 1 + 2 * x2, sd = exp(-1)),
                            y2 = rlnorm(nn, 1, sd = exp(-1 + x2)))
fit1 <- vglm(y1 ~ x2, lognormal(zero = 2), data = ldata2, trace = TRUE)
fit2 <- vglm(y2 ~ x2, lognormal(zero = 1), data = ldata2, trace = TRUE)
coef(fit1, matrix = TRUE)
coef(fit2, matrix = TRUE)
```

---

logofflink — *Log Link Function with an Offset*

Description

Computes the log transformation with an offset, including its inverse and the first two derivatives.

Usage

```
logofflink(theta, offset = 0, inverse = FALSE, deriv = 0,
           short = TRUE, tag = FALSE)
log1plink(theta, offset = 0, inverse = FALSE, deriv = 0,
          short = TRUE, tag = FALSE)
```

Arguments

theta	Numeric or character. See below for further details.
offset	Offset value. See Links. For log1plink this argument should not be used because the offset is implicitly unity .
inverse, deriv, short, tag	Details at Links.

Details

The log-offset link function is very commonly used for parameters that are greater than a certain value. In particular, it is defined by `log(theta + offset)` where `offset` is the offset value. For example, if `offset = 0.5` then the value of `theta` is restricted to be greater than -0.5.

Numerical values of `theta` close to `-offset` or out of range result in `Inf`, `-Inf`, `NA` or `NaN`.

The offset is implicitly 1 in `log1plink`. It is equivalent to `logofflink(offset = 1)` but is more accurate if `abs(theta)` is tiny. It may be used for `lrho` in `extbetabinomial` provided an offset `log(size - 1)` for η_2 is included.

Value

For `deriv = 0`, the log of `theta+offset`, i.e., `log(theta+offset)` when `inverse = FALSE`, and if `inverse = TRUE` then `exp(theta)-offset`.

For `deriv = 1`, then the function returns *d* `theta` / *d* `eta` as a function of `theta` if `inverse = FALSE`, else if `inverse = TRUE` then it returns the reciprocal.

Here, all logarithms are natural logarithms, i.e., to base *e*.

Note

The default means this function is identical to `loglink`.

Numerical instability may occur when `theta` is close to `-offset`.

Author(s)

Thomas W. Yee

References

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*, 2nd ed. London: Chapman & Hall.

See Also

`Links`, `loglink`, `extbetabinomial`.

Examples

```
## Not run:
logofflink(seq(-0.2, 0.5, by = 0.1))
logofflink(seq(-0.2, 0.5, by = 0.1), offset = 0.5)
       log(seq(-0.2, 0.5, by = 0.1) + 0.5)
## End(Not run)
```

Lomax *The Lomax Distribution*

Description

Density, distribution function, quantile function and random generation for the Lomax distribution with scale parameter `scale` and shape parameter `q`.

Usage

```
dlomax(x, scale = 1, shape3.q, log = FALSE)
plomax(q, scale = 1, shape3.q, lower.tail = TRUE, log.p = FALSE)
qlomax(p, scale = 1, shape3.q, lower.tail = TRUE, log.p = FALSE)
rlomax(n, scale = 1, shape3.q)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. If `length(n)` > 1, the length is taken to be the number required.
`scale`	scale parameter.
`shape3.q`	shape parameter.
`log`	Logical. If `log = TRUE` then the logarithm of the density is returned.
`lower.tail, log.p`	Same meaning as in `pnorm` or `qnorm`.

Details

See `lomax`, which is the **VGAM** family function for estimating the parameters by maximum likelihood estimation.

Value

`dlomax` gives the density, `plomax` gives the distribution function, `qlomax` gives the quantile function, and `rlomax` generates random deviates.

Note

The Lomax distribution is a special case of the 4-parameter generalized beta II distribution.

Author(s)

T. W. Yee and Kai Huang

References

Kleiber, C. and Kotz, S. (2003). *Statistical Size Distributions in Economics and Actuarial Sciences*, Hoboken, NJ, USA: Wiley-Interscience.

See Also

lomax, genbetaII.

Examples

```
probs <- seq(0.1, 0.9, by = 0.1)
max(abs(plomax(qlomax(p = probs, shape3.q =  1),
               shape3.q = 1) - probs))  # Should be 0

## Not run:  par(mfrow = c(1, 2))
x <- seq(-0.01, 5, len = 401)
plot(x, dexp(x), type = "l", col = "black", ylab = "", ylim = c(0, 3),
     main = "Black is std exponential, others are dlomax(x, shape3.q)")
lines(x, dlomax(x, shape3.q = 1), col = "orange")
lines(x, dlomax(x, shape3.q = 2), col = "blue")
lines(x, dlomax(x, shape3.q = 5), col = "green")
legend("topright", col = c("orange","blue","green"), lty = rep(1, 3),
       legend = paste("shape3.q =", c(1, 2, 5)))

plot(x, pexp(x), type = "l", col = "black", ylab = "", las = 1,
     main = "Black is std exponential, others are plomax(x, shape3.q)")
lines(x, plomax(x, shape3.q = 1), col = "orange")
lines(x, plomax(x, shape3.q = 2), col = "blue")
lines(x, plomax(x, shape3.q = 5), col = "green")
legend("bottomright", col = c("orange","blue","green"), lty = rep(1, 3),
       legend = paste("shape3.q =", c(1, 2, 5)))

## End(Not run)
```

---

lomax *Lomax Distribution Family Function*

Description

Maximum likelihood estimation of the 2-parameter Lomax distribution.

Usage

```
lomax(lscale = "loglink", lshape3.q = "loglink", iscale = NULL,
      ishape3.q = NULL, imethod = 1, gscale = exp(-5:5),
      gshape3.q = seq(0.75, 4, by = 0.25),
      probs.y = c(0.25, 0.5, 0.75), zero = "shape")
```

Arguments

`lscale, lshape3.q`
: Parameter link function applied to the (positive) parameters `scale` and `q`. See `Links` for more choices.

`iscale, ishape3.q, imethod`
: See `CommonVGAMffArguments` for information. For `imethod = 2` a good initial value for `iscale` is needed to obtain a good estimate for the other parameter.

`gscale, gshape3.q, zero, probs.y`
: See `CommonVGAMffArguments`.

Details

The 2-parameter Lomax distribution is the 4-parameter generalized beta II distribution with shape parameters $a = p = 1$. It is probably more widely known as the Pareto (II) distribution. It is also the 3-parameter Singh-Maddala distribution with shape parameter $a = 1$, as well as the beta distribution of the second kind with $p = 1$. More details can be found in Kleiber and Kotz (2003).

The Lomax distribution has density

$$f(y) = q/[b\{1 + y/b\}^{1+q}]$$

for $b > 0$, $q > 0$, $y \geq 0$. Here, b is the scale parameter `scale`, and `q` is a shape parameter. The cumulative distribution function is

$$F(y) = 1 - [1 + (y/b)]^{-q}.$$

The mean is

$$E(Y) = b/(q - 1)$$

provided $q > 1$; these are returned as the fitted values. This family function handles multiple responses.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Note

See the notes in `genbetaII`.

Author(s)

T. W. Yee

References

Kleiber, C. and Kotz, S. (2003). *Statistical Size Distributions in Economics and Actuarial Sciences*, Hoboken, NJ, USA: Wiley-Interscience.

See Also

Lomax, genbetaII, betaII, dagum, sinmad, fisk, inv.lomax, paralogistic, inv.paralogistic, simulate.vlm.

Examples

```
ldata <- data.frame(y = rlomax(n = 1000, scale =  exp(1), exp(2)))
fit <- vglm(y ~ 1, lomax, data = ldata, trace = TRUE)
coef(fit, matrix = TRUE)
Coef(fit)
summary(fit)
```

---

lpossums *Leadbeater's Possums*

Description

Abundance of Leadbeater's Possums observed in the field.

Usage

```
data(lpossums)
```

Format

A data frame with the following variables.

number Values between 0 and 10 excluding 6.

ofreq Observed frequency, i.e., the number of sites.

Details

A small data set recording the abundance of Leadbeater's Possums *Gymnobelideus leadbeateri* observed in the montane ash forests of the Central Highlands of Victoria, in south-eastern Australia. There are 151 3-hectare sites. The data has more 0s than usual relative to the Poisson, as well as exhibiting overdispersion too.

Source

Welsh, A. H., Cunningham, R. B., Donnelly, C. F. and Lindenmayer, D. B. (1996). Modelling the abundances of rare species: statistical models for counts with extra zeros. *Ecological Modelling*, **88**, 297–308.

See Also

zipoissonff.

Examples

```
lpossums
(samplemean <- with(lpossums, weighted.mean(number, ofreq)))
with(lpossums,  var(rep(number, times = ofreq)) / samplemean)
sum(with(lpossums, ofreq))
## Not run:  spikeplot(with(lpossums, rep(number, times = ofreq)),
  main = "Leadbeater's possums", col = "blue", xlab = "Number")
## End(Not run)
```

---

lqnorm *Minimizing the L-q norm Family Function*

Description

Minimizes the L-q norm of residuals in a linear model.

Usage

```
lqnorm(qpower = 2, link = "identitylink",
       imethod = 1, imu = NULL, ishrinkage = 0.95)
```

Arguments

qpower	A single numeric, must be greater than one, called q below. The absolute value of residuals are raised to the power of this argument, and then summed. This quantity is minimized with respect to the regression coefficients.
link	Link function applied to the 'mean' μ. See Links for more details.
imethod	Must be 1, 2 or 3. See CommonVGAMffArguments for more information. Ignored if imu is specified.
imu	Numeric, optional initial values used for the fitted values. The default is to use imethod = 1.
ishrinkage	How much shrinkage is used when initializing the fitted values. The value must be between 0 and 1 inclusive, and a value of 0 means the individual response values are used, and a value of 1 means the median or mean is used. This argument is used in conjunction with imethod = 3.

Details

This function minimizes the objective function

$$\sum_{i=1}^{n} w_i(|y_i - \mu_i|)^q$$

where q is the argument qpower, $\eta_i = g(\mu_i)$ where g is the link function, and η_i is the vector of linear/additive predictors. The prior weights w_i can be inputted using the weights argument of vlm/vglm/vgam etc.; it should be just a vector here since this function handles only a single vector or one-column response.

Numerical problem will occur when q is too close to one. Probably reasonable values range from 1.5 and up, say. The value $q = 2$ corresponds to ordinary least squares while $q = 1$ corresponds to the MLE of a double exponential (Laplace) distibution. The procedure becomes more sensitive to outliers the larger the value of q.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Warning

Convergence failure is common, therefore the user is advised to be cautious and monitor convergence!

Note

This **VGAM** family function is an initial attempt to provide a more robust alternative for regression and/or offer a little more flexibility than least squares. The @misc slot of the fitted object contains a list component called objectiveFunction which is the value of the objective function at the final iteration.

Author(s)

Thomas W. Yee

References

Yee, T. W. and Wild, C. J. (1996). Vector generalized additive models. *Journal of the Royal Statistical Society, Series B, Methodological*, **58**, 481–493.

See Also

uninormal.

Examples

```
set.seed(123)
ldata <- data.frame(x = sort(runif(nn <- 10 )))
realfun <- function(x) 4 + 5*x
ldata <- transform(ldata, y = realfun(x) + rnorm(nn, sd = exp(-1)))
# Make the first observation an outlier
ldata <- transform(ldata, y = c(4*y[1], y[-1]), x = c(-1, x[-1]))
fit <- vglm(y ~ x, lqnorm(qpower = 1.2), data = ldata)
coef(fit, matrix = TRUE)
head(fitted(fit))
fit@misc$qpower
fit@misc$objectiveFunction
```

```
## Not run:
# Graphical check
with(ldata, plot(x, y,
     main = paste0("LS = red, lqnorm = blue (qpower = ",
     fit@misc$qpower, "), truth = black"), col = "blue"))
lmfit <- lm(y ~ x, data = ldata)
with(ldata, lines(x,  fitted(fit), col = "blue"))
with(ldata, lines(x, lmfit$fitted, col = "red"))
with(ldata, lines(x,  realfun(x),  col = "black"))
## End(Not run)
```

`lrt.stat` *Likelihood Ratio Test Statistics Evaluated at the Null Values*

Description

Generic function that computes likelihood ratio test (LRT) statistics evaluated at the null values (consequently they do not suffer from the Hauck-Donner effect).

Usage

```
lrt.stat(object, ...)
lrt.stat.vlm(object, values0 = 0, subset = NULL, omit1s = TRUE,
           all.out = FALSE, trace = FALSE, ...)
```

Arguments

`object, values0, subset`
: Same as in `wald.stat.vlm`.

`omit1s, all.out, trace`
: Same as in `wald.stat.vlm`.

`...`
: Ignored for now.

Details

When `summary()` is applied to a `vglm` object a 4-column Wald table is produced. The corresponding p-values are generally viewed as inferior to those from a likelihood ratio test (LRT). For example, the Hauck and Donner (1977) effect (HDE) produces p-values that are biased upwards (see `hdeff`). Other reasons are that the Wald test is often less accurate (especially in small samples) and is not invariant to parameterization. By default, this function returns p-values based on the LRT by deleting one column at a time from the big VLM matrix and then restarting IRLS to obtain convergence (hopefully). Twice the difference between the log-likelihoods (or equivalently, the difference in the deviances if they are defined) is asymptotically chi-squared with 1 degree of freedom. One might expect the p-values from this function therefore to be more accurate and not suffer from the HDE. Thus this function is a recommended alternative (if it works) to `summaryvglm` for testing for the significance of a regression coefficient.

Value

By default, a vector of signed square root of the LRT statistics; these are asymptotically standard normal under the null hypotheses. If `all.out = TRUE` then a list is returned with the following components: `lrt.stat` the signed LRT statistics, `pvalues` the 2-sided p-values, `Lrt.stat2` the usual LRT statistic, `values0` the null values.

Warning

See `wald.stat.vlm`.

Author(s)

T. W. Yee.

See Also

`score.stat`, `wald.stat`, `summaryvglm`, `anova.vglm`, `vglm`, `lrtest`, `confintvglm`, `pchisq`, `profilevglm`, `hdeff`.

Examples

```
set.seed(1)
pneumo <- transform(pneumo, let = log(exposure.time),
                            x3 = rnorm(nrow(pneumo)))
fit <- vglm(cbind(normal, mild, severe) ~ let, propodds, pneumo)
cbind(coef(summary(fit)),
      "signed LRT stat" = lrt.stat(fit, omit1s = FALSE))
summary(fit, lrt0 = TRUE)  # Easy way to get it
```

`lrtest` *Likelihood Ratio Test of Nested Models*

Description

`lrtest` is a generic function for carrying out likelihood ratio tests. The default method can be employed for comparing nested VGLMs (see details below).

Usage

```
lrtest(object, ...)

lrtest_vglm(object, ..., no.warning = FALSE, name = NULL)
```

Arguments

`object`	a `vglm` object. See below for details.
`...`	further object specifications passed to methods. See below for details.
`no.warning`	logical; if `TRUE` then no warning is issued. For example, setting `TRUE` might be a good idea when testing for linearity of a variable for a `"pvgam"` object.
`name`	a function for extracting a suitable name/description from a fitted model object. By default the name is queried by calling `formula`.

Details

`lrtest` is intended to be a generic function for comparisons of models via asymptotic likelihood ratio tests. The default method consecutively compares the fitted model object `object` with the models passed in `...`. Instead of passing the fitted model objects in `...`, several other specifications are possible. The updating mechanism is the same as for `waldtest()` in **lmtest**: the models in `...` can be specified as integers, characters (both for terms that should be eliminated from the previous model), update formulas or fitted model objects. Except for the last case, the existence of an `update` method is assumed. See `waldtest()` in **lmtest** for details.

Subsequently, an asymptotic likelihood ratio test for each two consecutive models is carried out: Twice the difference in log-likelihoods (as derived by the `logLik` methods) is compared with a Chi-squared distribution.

Value

An object of class `"VGAManova"` which contains a slot with the log-likelihood, degrees of freedom, the difference in degrees of freedom, likelihood ratio Chi-squared statistic and corresponding p value. These are printed by `stats:::print.anova()`; see `anova`.

Warning

Several **VGAM** family functions implement distributions which do not satisfying the usual regularity conditions needed for the LRT to work. No checking or warning is given for these.

Note

The code was adapted directly from **lmtest** (written by T. Hothorn, A. Zeileis, G. Millo, D. Mitchell) and made to work for VGLMs and S4. This help file also was adapted from **lmtest**.

Approximate LRTs might be applied to VGAMs, as produced by `vgam`, but it is probably better in inference to use `vglm` with regression splines (`bs` and `ns`). This methods function should not be applied to other models such as those produced by `rrvglm`, by `cqo`, by `cao`.

See Also

lmtest, `vglm`, `lrt.stat.vlm`, `score.stat.vlm`, `wald.stat.vlm`, `anova.vglm`.

Examples

```
set.seed(1)
pneumo <- transform(pneumo, let = log(exposure.time),
                            x3  = runif(nrow(pneumo)))
fit1 <- vglm(cbind(normal, mild, severe) ~ let     , propodds, pneumo)
fit2 <- vglm(cbind(normal, mild, severe) ~ let + x3, propodds, pneumo)
fit3 <- vglm(cbind(normal, mild, severe) ~ let     , cumulative, pneumo)
# Various equivalent specifications of the LR test for testing x3
(ans1 <- lrtest(fit2, fit1))
ans2 <- lrtest(fit2, 2)
ans3 <- lrtest(fit2, "x3")
ans4 <- lrtest(fit2, . ~ . - x3)
c(all.equal(ans1, ans2), all.equal(ans1, ans3), all.equal(ans1, ans4))

# Doing it manually
(testStatistic <- 2 * (logLik(fit2) - logLik(fit1)))
(pval <- pchisq(testStatistic, df = df.residual(fit1) - df.residual(fit2),
                lower.tail = FALSE))

(ans4 <- lrtest(fit3, fit1))  # Test PO (parallelism) assumption
```

---

lvplot *Latent Variable Plot*

Description

Generic function for a *latent variable plot* (also known as an *ordination diagram* by ecologists).

Usage

```
lvplot(object, ...)
```

Arguments

`object`	An object for a latent variable plot is meaningful.
`...`	Other arguments fed into the specific methods function of the model. They usually are graphical parameters, and sometimes they are fed into the methods function for `Coef`.

Details

Latent variables occur in reduced-rank regression models, as well as in quadratic and additive ordination. For the latter, latent variables are often called the *site scores*. Latent variable plots were coined by Yee (2004), and have the latent variable as at least one of its axes.

Value

The value returned depends specifically on the methods function invoked.

Note

Latent variables are not really applicable to vglm/vgam models.

Author(s)

Thomas W. Yee

References

Yee, T. W. (2004). A new technique for maximum-likelihood canonical Gaussian ordination. *Ecological Monographs*, **74**, 685–701.

Yee, T. W. (2006). Constrained additive ordination. *Ecology*, **87**, 203–213.

See Also

lvplot.qrrvglm, lvplot.cao, latvar, trplot.

Examples

```
## Not run:
hspider[,1:6] <- scale(hspider[,1:6])  # Stdz environmental vars
set.seed(123)
p1 <- cao(cbind(Pardlugu, Pardmont, Pardnigr, Pardpull, Zoraspin) ~
          WaterCon + BareSand + FallTwig +
          CoveMoss + CoveHerb + ReflLux,
          family = poissonff, data = hspider, Bestof = 3,
          df1.nl = c(Zoraspin = 2.5, 3), Crow1positive = TRUE)
index <- 1:ncol(depvar(p1))
lvplot(p1, lcol = index, pcol = index, y = TRUE, las = 1)

## End(Not run)
```

---

lvplot.qrrvglm *Latent Variable Plot for QO models*

Description

Produces an ordination diagram (latent variable plot) for quadratic ordination (QO) models. For rank-1 models, the x-axis is the first ordination/constrained/canonical axis. For rank-2 models, the x- and y-axis are the first and second ordination axes respectively.

Usage

```
lvplot.qrrvglm(object, varI.latvar = FALSE, refResponse = NULL,
    add = FALSE, show.plot = TRUE,
    rug = TRUE, y = FALSE, type = c("fitted.values", "predictors"),
    xlab = paste0("Latent Variable", if (Rank == 1) "" else " 1"),
    ylab = if (Rank == 1) switch(type, predictors = "Predictors",
```

```
fitted.values = "Fitted values") else "Latent Variable 2",
pcex = par()$cex, pcol = par()$col, pch = par()$pch,
llty = par()$lty, lcol = par()$col, llwd = par()$lwd,
label.arg = FALSE, adj.arg = -0.1,
ellipse = 0.95, Absolute = FALSE, elty = par()$lty,
ecol = par()$col, elwd = par()$lwd, egrid = 200,
chull.arg = FALSE, clty = 2, ccol = par()$col, clwd = par()$lwd,
cpch = "   ",
C = FALSE, OriginC = c("origin", "mean"),
Clty = par()$lty, Ccol = par()$col, Clwd = par()$lwd,
Ccex = par()$cex, Cadj.arg = -0.1, stretchC = 1,
sites = FALSE, spch = NULL, scol = par()$col, scex = par()$cex,
sfont = par()$font, check.ok = TRUE, jitter.y = FALSE, ...)
```

Arguments

`object`	A CQO object.
`varI.latvar`	Logical that is fed into `Coef.qrrvglm`.
`refResponse`	Integer or character that is fed into `Coef.qrrvglm`.
`add`	Logical. Add to an existing plot? If `FALSE`, a new plot is made.
`show.plot`	Logical. Plot it?
`rug`	Logical. If `TRUE`, a rug plot is plotted at the foot of the plot (applies to rank-1 models only). These values are jittered to expose ties.
`y`	Logical. If `TRUE`, the responses will be plotted (applies only to rank-1 models and if `type = "fitted.values"`.)
`type`	Either `"fitted.values"` or `"predictors"`, specifies whether the y-axis is on the response or eta-scales respectively.
`xlab`	Caption for the x-axis. See `par`.
`ylab`	Caption for the y-axis. See `par`.
`pcex`	Character expansion of the points. Here, for rank-1 models, points are the response y data. For rank-2 models, points are the optimums. See the `cex` argument in `par`.
`pcol`	Color of the points. See the `col` argument in `par`.
`pch`	Either an integer specifying a symbol or a single character to be used as the default in plotting points. See `par`. The `pch` argument can be of length M, the number of species.
`llty`	Line type. Rank-1 models only. See the `lty` argument of `par`.
`lcol`	Line color. Rank-1 models only. See the `col` argument of `par`.
`llwd`	Line width. Rank-1 models only. See the `lwd` argument of `par`.
`label.arg`	Logical. Label the optimums and **C**? (applies only to rank-2 models only).
`adj.arg`	Justification of text strings for labelling the optimums (applies only to rank-2 models only). See the `adj` argument of `par`.

`ellipse`	Numerical, of length 0 or 1 (applies only to rank-2 models only). If `Absolute` is `TRUE` then `ellipse` should be assigned a value that is used for the elliptical contouring. If `Absolute` is `FALSE` then `ellipse` should be assigned a value between 0 and 1, for example, setting `ellipse = 0.9` means an ellipse with contour = 90% of the maximum will be plotted about each optimum. If `ellipse` is a negative value, then the function checks that the model is an equal-tolerances model and `varI.latvar = FALSE`, and if so, plots circles with radius `-ellipse`. For example, setting `ellipse = -1` will result in circular contours that have unit radius (in latent variable units). If `ellipse` is `NULL` or `FALSE` then no ellipse is drawn around the optimums.
`Absolute`	Logical. If `TRUE`, the contours corresponding to `ellipse` are on an absolute scale. If `FALSE`, the contours corresponding to `ellipse` are on a relative scale.
`elty`	Line type of the ellipses. See the `lty` argument of `par`.
`ecol`	Line color of the ellipses. See the `col` argument of `par`.
`elwd`	Line width of the ellipses. See the `lwd` argument of `par`.
`egrid`	Numerical. Line resolution of the ellipses. Choosing a larger value will result in smoother ellipses. Useful when ellipses are large.
`chull.arg`	Logical. Add a convex hull around the site scores?
`clty`	Line type of the convex hull. See the `lty` argument of `par`.
`ccol`	Line color of the convex hull. See the `col` argument of `par`.
`clwd`	Line width of the convex hull. See the `lwd` argument of `par`.
`cpch`	Character to be plotted at the intersection points of the convex hull. Having white spaces means that site labels are not obscured there. See the `pch` argument of `par`.
`C`	Logical. Add **C** (represented by arrows emanating from `OriginC`) to the plot?
`OriginC`	Character or numeric. Where the arrows representing **C** emanate from. If character, it must be one of the choices given. By default the first is chosen. The value `"origin"` means `c(0,0)`. The value `"mean"` means the sample mean of the latent variables (centroid). Alternatively, the user may specify a numerical vector of length 2.
`Clty`	Line type of the arrows representing **C**. See the `lty` argument of `par`.
`Ccol`	Line color of the arrows representing **C**. See the `col` argument of `par`.
`Clwd`	Line width of the arrows representing **C**. See the `lwd` argument of `par`.
`Ccex`	Numeric. Character expansion of the labelling of **C**. See the `cex` argument of `par`.
`Cadj.arg`	Justification of text strings when labelling **C**. See the `adj` argument of `par`.
`stretchC`	Numerical. Stretching factor for **C**. Instead of using **C**, `stretchC *` **C** is used.
`sites`	Logical. Add the site scores (aka latent variable values, nu's) to the plot? (applies only to rank-2 models only).
`spch`	Plotting character of the site scores. The default value of `NULL` means the row labels of the data frame are used. They often are the site numbers. See the `pch` argument of `par`.

`scol`	Color of the site scores. See the `col` argument of `par`.
`scex`	Character expansion of the site scores. See the `cex` argument of `par`.
`sfont`	Font used for the site scores. See the `font` argument of `par`.
`check.ok`	Logical. Whether a check is performed to see that `noRRR = ~ 1` was used. It doesn't make sense to have a latent variable plot unless this is so.
`jitter.y`	Logical. If `y` is plotted, jitter it first? This may be useful for counts and proportions.
`...`	Arguments passed into the `plot` function when setting up the entire plot. Useful arguments here include `xlim` and `ylim`.

Details

This function only works for rank-1 and rank-2 QRR-VGLMs with argument `noRRR = ~ 1`.

For unequal-tolerances models, the latent variable axes can be rotated so that at least one of the tolerance matrices is diagonal; see `Coef.qrrvglm` for details.

Arguments beginning with "`p`" correspond to the points e.g., `pcex` and `pcol` correspond to the size and color of the points. Such "`p`" arguments should be vectors of length 1, or n, the number of sites. For the rank-2 model, arguments beginning with "`p`" correspond to the optimums.

Value

Returns a matrix of latent variables (site scores) regardless of whether a plot was produced or not.

Warning

Interpretation of a latent variable plot (CQO diagram) is potentially very misleading in terms of distances if (i) the tolerance matrices of the species are unequal and (ii) the contours of these tolerance matrices are not included in the ordination diagram.

Note

A species which does not have an optimum will not have an ellipse drawn even if requested, i.e., if its tolerance matrix is not positive-definite.

Plotting **C** gives a visual display of the weights (loadings) of each of the variables used in the linear combination defining each latent variable.

The arguments `elty`, `ecol` and `elwd`, may be replaced in the future by `llty`, `lcol` and `llwd`, respectively.

For rank-1 models, a similar function to this one is `perspqrrvglm`. It plots the fitted values on a more fine grid rather than at the actual site scores here. The result is a collection of smooth bell-shaped curves. However, it has the weakness that the plot is more divorced from the data; the user thinks it is the truth without an appreciation of the statistical variability in the estimates.

In the example below, the data comes from an equal-tolerances model. The species' tolerance matrices are all the identity matrix, and the optimums are at (0,0), (1,1) and (-2,0) for species 1, 2, 3 respectively.

Author(s)

Thomas W. Yee

References

Yee, T. W. (2004). A new technique for maximum-likelihood canonical Gaussian ordination. *Ecological Monographs*, **74**, 685–701.

See Also

lvplot, perspqrrvglm, Coef.qrrvglm, par, cqo.

Examples

```
set.seed(123); nn <- 200
cdata <- data.frame(x2 = rnorm(nn),  # Mean 0 (needed when I.tol=TRUE)
                    x3 = rnorm(nn),  # Mean 0 (needed when I.tol=TRUE)
                    x4 = rnorm(nn))  # Mean 0 (needed when I.tol=TRUE)
cdata <- transform(cdata, latvar1 =  x2 + x3 - 2*x4,
                          latvar2 = -x2 + x3 + 0*x4)
# Nb. latvar2 is weakly correlated with latvar1
cdata <- transform(cdata,
           lambda1 = exp(6 - 0.5 * (latvar1-0)^2 - 0.5 * (latvar2-0)^2),
           lambda2 = exp(5 - 0.5 * (latvar1-1)^2 - 0.5 * (latvar2-1)^2),
           lambda3 = exp(5 - 0.5 * (latvar1+2)^2 - 0.5 * (latvar2-0)^2))
cdata <- transform(cdata,
            spp1 = rpois(nn, lambda1),
            spp2 = rpois(nn, lambda2),
            spp3 = rpois(nn, lambda3))
set.seed(111)
## Not run:
p2 <- cqo(cbind(spp1, spp2, spp3) ~ x2 + x3 + x4, poissonff,
          data = cdata, Rank = 2, I.tolerances = TRUE,
          Crow1positive = c(TRUE, FALSE))  # deviance = 505.81
if (deviance(p2) > 506) stop("suboptimal fit obtained")
sort(deviance(p2, history = TRUE))  # A history of the iterations
Coef(p2)

## End(Not run)

## Not run:
lvplot(p2, sites = TRUE, spch = "*", scol = "darkgreen", scex = 1.5,
  chull = TRUE, label = TRUE, Absolute = TRUE, ellipse = 140,
  adj = -0.5, pcol = "blue", pcex = 1.3, las = 1, Ccol = "orange",
  C = TRUE, Cadj = c(-0.3, -0.3, 1), Clwd = 2, Ccex = 1.4,
  main = paste("Contours at Abundance = 140 with",
               "convex hull of the site scores"))
## End(Not run)
## Not run:
var(latvar(p2))  # A diagonal matrix, i.e., uncorrelated latent vars
var(latvar(p2, varI.latvar = TRUE))  # Identity matrix
Tol(p2)[, , 1:2]  # Identity matrix
```

```
Tol(p2, varI.latvar = TRUE)[, , 1:2]  # A diagonal matrix

## End(Not run)
```

`lvplot.rrvglm` *Latent Variable Plot for RR-VGLMs*

Description

Produces an *ordination diagram* (also known as a *biplot* or *latent variable plot*) for *reduced-rank vector generalized linear models* (RR-VGLMs). For rank-2 models only, the x- and y-axis are the first and second canonical axes respectively.

Usage

```
lvplot.rrvglm(object,
    A = TRUE, C = TRUE, scores = FALSE, show.plot = TRUE,
    groups = rep(1, n), gapC = sqrt(sum(par()$cxy^2)),
    scaleA = 1,
    xlab = "Latent Variable 1", ylab = "Latent Variable 2",
    Alabels = if (length(object@misc$predictors.names))
    object@misc$predictors.names else param.names("LP", M),
    Aadj = par()$adj, Acex = par()$cex, Acol = par()$col,
    Apch = NULL,
    Clabels = rownames(Cmat), Cadj = par()$adj,
    Ccex = par()$cex, Ccol = par()$col, Clty = par()$lty,
    Clwd = par()$lwd,
    chull.arg = FALSE, ccex = par()$cex, ccol = par()$col,
    clty = par()$lty, clwd = par()$lwd,
    spch = NULL, scex = par()$cex, scol = par()$col,
    slabels = rownames(x2mat), ...)
```

Arguments

`object`	Object of class `"rrvglm"`.
`A`	Logical. Allow the plotting of **A**?
`C`	Logical. Allow the plotting of **C**? If `TRUE` then **C** is represented by arrows emenating from the origin.
`scores`	Logical. Allow the plotting of the n scores? The scores are the values of the latent variables for each observation.
`show.plot`	Logical. Plot it? If `FALSE`, no plot is produced and the matrix of scores (n latent variable values) is returned. If `TRUE`, the rank of `object` need not be 2.
`groups`	A vector whose distinct values indicate which group the observation belongs to. By default, all the observations belong to a single group. Useful for the multinomial logit model (see `multinomial`.

`gapC`	The gap between the end of the arrow and the text labelling of **C**, in latent variable units.
`scaleA`	Numerical value that is multiplied by **A**, so that **C** is divided by this value.
`xlab`	Caption for the x-axis. See `par`.
`ylab`	Caption for the y-axis. See `par`.
`Alabels`	Character vector to label **A**. Must be of length M.
`Aadj`	Justification of text strings for labelling **A**. See the `adj` argument of `par`.
`Acex`	Numeric. Character expansion of the labelling of **A**. See the `cex` argument of `par`.
`Acol`	Line color of the arrows representing **C**. See the `col` argument of `par`.
`Apch`	Either an integer specifying a symbol or a single character to be used as the default in plotting points. See `par`. The `pch` argument can be of length M, the number of species.
`Clabels`	Character vector to label **C**. Must be of length $p2$.
`Cadj`	Justification of text strings for labelling **C**. See the `adj` argument of `par`.
`Ccex`	Numeric. Character expansion of the labelling of **C**. See the `cex` argument of `par`.
`Ccol`	Line color of the arrows representing **C**. See the `col` argument of `par`.
`Clty`	Line type of the arrows representing **C**. See the `lty` argument of `par`.
`Clwd`	Line width of the arrows representing **C**. See the `lwd` argument of `par`.
`chull.arg`	Logical. Plot the convex hull of the scores? This is done for each group (see the `group` argument).
`ccex`	Numeric. Character expansion of the labelling of the convex hull. See the `cex` argument of `par`.
`ccol`	Line color of the convex hull. See the `col` argument of `par`.
`clty`	Line type of the convex hull. See the `lty` argument of `par`.
`clwd`	Line width of the convex hull. See the `lwd` argument of `par`.
`spch`	Either an integer specifying a symbol or a single character to be used as the default in plotting points. See `par`. The `spch` argument can be of length M, number of species.
`scex`	Numeric. Character expansion of the labelling of the scores. See the `cex` argument of `par`.
`scol`	Line color of the arrows representing **C**. See the `col` argument of `par`.
`slabels`	Character vector to label the scores. Must be of length n.
`...`	Arguments passed into the `plot` function when setting up the entire plot. Useful arguments here include `xlim` and `ylim`.

Details

For RR-VGLMs, a *biplot* and a *latent variable* plot coincide. In general, many of the arguments starting with “A” refer to **A** (of length M), “C” to **C** (of length $p2$), “c” to the convex hull (of length `length(unique(groups))`), and “s” to scores (of length n).

As the result is a biplot, its interpretation is based on the inner product.

Value

The matrix of scores (n latent variable values) is returned regardless of whether a plot was produced or not.

Note

The functions lvplot.rrvglm and biplot.rrvglm are equivalent.

In the example below the predictor variables are centered, which is a good idea.

Author(s)

Thomas W. Yee

References

Yee, T. W. and Hastie, T. J. (2003). Reduced-rank vector generalized linear models. *Statistical Modelling*, **3**, 15–41.

See Also

lvplot, par, rrvglm, Coef.rrvglm, rrvglm.control.

Examples

```
 set.seed(1)
nn <- nrow(pneumo)  # x1--x3 are some unrelated covariates
pneumo <-
  transform(pneumo, slet = scale(log(exposure.time)),
                    imag = severe + 3,  # Fictitional!
                    x1 = rnorm(nn), x2 = rnorm(nn), x3 = rnorm(nn))
fit <-
  rrvglm(cbind(normal, mild, severe, imag) ~ slet + x1 + x2 + x3,
#             Corner = FALSE, Uncorrel = TRUE,   # orig.
              multinomial, data = pneumo, Rank = 2)
## Not run:
lvplot(fit, chull = TRUE, scores = TRUE, clty = 2, ccol = 4,
       scol = "red", Ccol = "green3", Clwd = 2, Ccex = 2,
       main = "Biplot of some fictitional data")
## End(Not run)
```

---

machinists *Machinists Accidents*

Description

A small count data set involving 414 machinists from a three months study, of accidents around the end of WWI.

Usage

```
data(machinists)
```

Format

A data frame with the following variables.

accidents The number of accidents

ofreq Observed frequency, i.e., the number of machinists with that many accidents

Details

The data was collected over a period of three months. There were 414 machinists in total. Also, there were data collected over six months, but it is not given here.

Source

Incidence of Industrial Accidents. Report No. 4 (Industrial Fatigue Research Board), Stationery Office, London, 1919.

References

Greenwood, M. and Yule, G. U. (1920). An Inquiry into the Nature of Frequency Distributions Representative of Multiple Happenings with Particular Reference to the Occurrence of Multiple Attacks of Disease or of Repeated Accidents. *Journal of the Royal Statistical Society*, **83**, 255–279.

See Also

negbinomial, poissonff.

Examples

```
machinists
mean(with(machinists, rep(accidents, times = ofreq)))
 var(with(machinists, rep(accidents, times = ofreq)))
## Not run:  barplot(with(machinists, ofreq),
          names.arg = as.character(with(machinists, accidents)),
          main = "Machinists accidents",
          col = "lightblue", las = 1,
          ylab = "Frequency", xlab = "accidents")
## End(Not run)
```

Makeham *The Makeham Distribution*

Description

Density, cumulative distribution function, quantile function and random generation for the Makeham distribution.

Usage

```
dmakeham(x, scale = 1, shape, epsilon = 0, log = FALSE)
pmakeham(q, scale = 1, shape, epsilon = 0, lower.tail = TRUE,
         log.p = FALSE)
qmakeham(p, scale = 1, shape, epsilon = 0, lower.tail = TRUE,
         log.p = FALSE)
rmakeham(n, scale = 1, shape, epsilon = 0)
```

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
n	number of observations. Same as in runif.
log	Logical. If log = TRUE then the logarithm of the density is returned.
lower.tail, log.p	Same meaning as in pnorm or qnorm.
scale, shape	positive scale and shape parameters.
epsilon	another parameter. Must be non-negative. See below.

Details

See makeham for details. The default value of epsilon = 0 corresponds to the Gompertz distribution. The function pmakeham uses lambertW.

Value

dmakeham gives the density, pmakeham gives the cumulative distribution function, qmakeham gives the quantile function, and rmakeham generates random deviates.

Author(s)

T. W. Yee and Kai Huang

References

Jodra, P. (2009). A closed-form expression for the quantile function of the Gompertz-Makeham distribution. *Mathematics and Computers in Simulation*, **79**, 3069–3075.

See Also

makeham, lambertW.

Examples

```
probs <- seq(0.01, 0.99, by = 0.01)
Shape <- exp(-1); Scale <- exp(1); eps = Epsilon <- exp(-1)
max(abs(pmakeham(qmakeham(probs, sca = Scale, Shape, eps = Epsilon),
    sca = Scale, Shape, eps = Epsilon) - probs))  # Should be 0

## Not run:  x <- seq(-0.1, 2.0, by = 0.01);
plot(x, dmakeham(x, sca = Scale, Shape, eps = Epsilon), type = "l",
     main = "Blue is density, orange is the CDF",
     sub = "Purple lines are the 10,20,...,90 percentiles",
     col = "blue", las = 1, ylab = "")
abline(h = 0, col = "blue", lty = 2)
lines(x, pmakeham(x, sca = Scale, Shape, eps = Epsilon), col = "orange")
probs <- seq(0.1, 0.9, by = 0.1)
Q <- qmakeham(probs, sca = Scale, Shape, eps = Epsilon)
lines(Q, dmakeham(Q, sca = Scale, Shape, eps = Epsilon),
      col = "purple", lty = 3, type = "h")
pmakeham(Q, sca = Scale, Shape, eps = Epsilon) - probs # Should be all 0
abline(h = probs, col = "purple", lty = 3)
## End(Not run)
```

---

makeham *Makeham Regression Family Function*

Description

Maximum likelihood estimation of the 3-parameter Makeham distribution.

Usage

```
makeham(lscale = "loglink", lshape = "loglink", lepsilon = "loglink",
        iscale = NULL,   ishape = NULL,   iepsilon = NULL,
        gscale = exp(-5:5),gshape = exp(-5:5), gepsilon = exp(-4:1),
        nsimEIM = 500, oim.mean = TRUE, zero = NULL, nowarning = FALSE)
```

Arguments

`nowarning` Logical. Suppress a warning? Ignored for **VGAM** 0.9-7 and higher.

`lshape, lscale, lepsilon`
Parameter link functions applied to the shape parameter `shape`, scale parameter `scale`, and other parameter `epsilon`. All parameters are treated as positive here (cf. `dmakeham` allows `epsilon = 0`, etc.). See `Links` for more choices.

`ishape, iscale, iepsilon`
Optional initial values. A `NULL` means a value is computed internally. A value must be given for `iepsilon` currently, and this is a sensitive parameter!

gshape, gscale, gepsilon
: See CommonVGAMffArguments.

nsimEIM, zero
: See CommonVGAMffArguments. Argument probs.y is used only when imethod = 2.

oim.mean
: To be currently ignored.

Details

The Makeham distribution, which adds another parameter to the Gompertz distribution, has cumulative distribution function

$$F(y; \alpha, \beta, \varepsilon) = 1 - \exp\left\{-y\varepsilon + \frac{\alpha}{\beta}\left[1 - e^{\beta y}\right]\right\}$$

which leads to a probability density function

$$f(y; \alpha, \beta, \varepsilon) = \left[\varepsilon + \alpha e^{\beta y}\right] \exp\left\{-y\varepsilon + \frac{\alpha}{\beta}\left[1 - e^{\beta y}\right]\right\},$$

for $\alpha > 0$, $\beta > 0$, $\varepsilon \geq 0$, $y > 0$. Here, β is called the scale parameter scale, and α is called a shape parameter. The moments for this distribution do not appear to be available in closed form.

Simulated Fisher scoring is used and multiple responses are handled.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Warning

A lot of care is needed because this is a rather difficult distribution for parameter estimation, especially when the shape parameter is large relative to the scale parameter. If the self-starting initial values fail then try experimenting with the initial value arguments, especially iepsilon. Successful convergence depends on having very good initial values. More improvements could be made here. Also, monitor convergence by setting trace = TRUE.

A trick is to fit a gompertz distribution and use it for initial values; see below. However, this family function is currently numerically fraught.

Author(s)

T. W. Yee

See Also

dmakeham, gompertz, simulate.vlm.

Examples

```
## Not run:  set.seed(123)
mdata <- data.frame(x2 = runif(nn <- 1000))
mdata <- transform(mdata, eta1  = -1,
                          ceta1 =  1,
                          eeta1 = -2)
mdata <- transform(mdata, shape1 = exp(eta1),
                          scale1 = exp(ceta1),
                          epsil1 = exp(eeta1))
mdata <- transform(mdata,
         y1 = rmakeham(nn, shape = shape1, scale = scale1, eps = epsil1))

# A trick is to fit a Gompertz distribution first
fit0 <- vglm(y1 ~ 1, gompertz, data = mdata, trace = TRUE)
fit1 <- vglm(y1 ~ 1, makeham, data = mdata,
             etastart = cbind(predict(fit0), log(0.1)), trace = TRUE)

coef(fit1, matrix = TRUE)
summary(fit1)

## End(Not run)
```

`margeff` *Marginal Effects for Several Categorical Response Models*

Description

Marginal effects for the multinomial logit model and cumulative logit/probit/... models and continuation ratio models and stopping ratio models and adjacent categories models: the derivative of the fitted probabilities with respect to each explanatory variable.

Usage

```
margeff(object, subset = NULL, ...)
```

Arguments

`object`	A `vglm` object, with one of the following family functions: `multinomial`, `cumulative`, `cratio`, `sratio`, `acat`, `poissonff`, `negbinomial` or `posnegbinomial`.
`subset`	Numerical or logical vector, denoting the required observation(s). Recycling is used if possible. The default means all observations.
`...`	further arguments passed into the other methods functions.

Details

Computes the derivative of the fitted probabilities of the categorical response model with respect to each explanatory variable. Formerly one big function, this function now uses S4 dispatch to break up the computations.

The function `margeff()` is *not* generic. However, it calls the function `margeffS4VGAM()` which *is*. This is based on the class of the `VGAMff` argument, and it uses the S4 function `setMethod` to correctly dispatch to the required methods function. The inheritance is given by the `vfamily` slot of the **VGAM** family function.

Value

A p by $M + 1$ by n array, where p is the number of explanatory variables and the (hopefully) nominal response has $M + 1$ levels, and there are n observations.

In general, if `is.numeric(subset)` and `length(subset) == 1` then a p by $M+1$ matrix is returned.

Warning

Care is needed in interpretation, e.g., the change is not universally accurate for a unit change in each explanatory variable because eventually the 'new' probabilities may become negative or greater than unity. Also, the 'new' probabilities will not sum to one.

This function is not applicable for models with data-dependent terms such as `bs` and `poly`. Also the function should not be applied to models with any terms that have generated more than one column of the LM model matrix, such as `bs` and `poly`. For such try using numerical methods such as finite-differences. The `formula` in `object` should comprise of simple terms of the form `~ x2 + x3 + x4`, etc.

Some numerical problems may occur if the fitted values are close to 0 or 1 for the `cratio` and `sratio` models. Models with offsets may result in an incorrect answer.

Note

For `multinomial` this function should handle any value of `refLevel` and also any constraint matrices. However, it does not currently handle the `xij` or `form2` arguments, nor `vgam` objects.

If marginal effects are to be computed for some values not equal to those used in the training set, then the `@x` and the `@predictors` slots both need to be assigned. See Example 3 below.

Some other limitations are imposed, e.g., for `acat` models only a `loglink` link is allowed.

Author(s)

T. W. Yee, with some help and motivation from Stasha Rmandic.

See Also

`multinomial`, `cumulative`, `propodds`, `acat`, `cratio`, `sratio`, `poissonff`, `negbinomial`, `vglm`.

Examples

```
# Not a good example for multinomial() since the response is ordinal!!
ii <- 3; hh <- 1/100
pneumo <- transform(pneumo, let = log(exposure.time))
fit <- vglm(cbind(normal, mild, severe) ~ let, multinomial, pneumo)
fit <- vglm(cbind(normal, mild, severe) ~ let,
            cumulative(reverse = TRUE,  parallel = TRUE),
            data = pneumo)
fitted(fit)[ii, ]

mynewdata <- with(pneumo, data.frame(let = let[ii] + hh))
(newp <- predict(fit, newdata = mynewdata, type = "response"))

# Compare the difference. Should be the same as hh --> 0.
round((newp-fitted(fit)[ii, ]) / hh, 3)  # Finite-diff approxn
round(margeff(fit, subset = ii)["let",], 3)

# Other examples
round(margeff(fit), 3)
round(margeff(fit, subset = 2)["let",], 3)
round(margeff(fit, subset = c(FALSE, TRUE))["let",,], 3)  # Recycling
round(margeff(fit, subset = c(2, 4, 6, 8))["let",,], 3)

# Example 3; margeffs at a new value
mynewdata2a <- data.frame(let = 2)  # New value
mynewdata2b <- data.frame(let = 2 + hh)  # For finite-diff approxn
(neweta2 <- predict(fit, newdata = mynewdata2a))
fit@x[1, ] <- c(1, unlist(mynewdata2a))
fit@predictors[1, ] <- neweta2  # Needed
max(abs(margeff(fit, subset = 1)["let", ] - (
        predict(fit, newdata = mynewdata2b, type = "response") -
        predict(fit, newdata = mynewdata2a, type = "response")) / hh
))  # Should be 0
```

`marital.nz` *New Zealand Marital Data*

Description

Some marital data mainly from a large NZ company collected in the early 1990s.

Usage

```
data(marital.nz)
```

Format

A data frame with 6053 observations on the following 3 variables.

age a numeric vector, age in years

`ethnicity` a factor with levels `European` `Maori` `Other` `Polynesian`. Only Europeans are included in the data set.

`mstatus` a factor with levels `Divorced/Separated`, `Married/Partnered`, `Single`, `Widowed`.

Details

This is a subset of a data set collected from a self-administered questionnaire administered in a large New Zealand workforce observational study conducted during 1992–3. The data were augmented by a second study consisting of retirees. The data can be considered a reasonable representation of the white male New Zealand population in the early 1990s.

Source

Clinical Trials Research Unit, University of Auckland, New Zealand.

References

See `bmi.nz` and `chest.nz`.

Examples

```
summary(marital.nz)
```

`Max` *Maximums*

Description

Generic function for the *maximums* (maxima) of a model.

Usage

```
Max(object, ...)
```

Arguments

`object`	An object for which the computation or extraction of a maximum (or maximums) is meaningful.
`...`	Other arguments fed into the specific methods function of the model. Sometimes they are fed into the methods function for `Coef`.

Details

Different models can define a maximum in different ways. Many models have no such notion or definition.

Maximums occur in quadratic and additive ordination, e.g., CQO or CAO. For these models the maximum is the fitted value at the optimum. For quadratic ordination models there is a formula for the optimum but for additive ordination models the optimum must be searched for numerically. If it occurs on the boundary, then the optimum is undefined. For a valid optimum, the fitted value at the optimum is the maximum.

Value

The value returned depends specifically on the methods function invoked.

Author(s)

Thomas W. Yee

References

Yee, T. W. (2004). A new technique for maximum-likelihood canonical Gaussian ordination. *Ecological Monographs*, **74**, 685–701.

Yee, T. W. (2006). Constrained additive ordination. *Ecology*, **87**, 203–213.

See Also

`Max.qrrvglm`, `Tol`, `Opt`.

Examples

```
## Not run:
set.seed(111)  # This leads to the global solution
hspider[,1:6] <- scale(hspider[,1:6])  # Standardized environmental vars
p1 <- cqo(cbind(Alopacce, Alopcune, Alopfabr, Arctlute, Arctperi,
                Auloalbi, Pardlugu, Pardmont, Pardnigr, Pardpull,
                Trocterr, Zoraspin) ~
          WaterCon + BareSand + FallTwig + CoveMoss + CoveHerb + ReflLux,
          poissonff, Bestof = 2, data = hspider, Crow1positive = FALSE)
Max(p1)

index <- 1:ncol(depvar(p1))
persp(p1, col = index, las = 1, llwd = 2)
abline(h = Max(p1), lty = 2, col = index)

## End(Not run)
```

`Maxwell` *The Maxwell Distribution*

Description

Density, distribution function, quantile function and random generation for the Maxwell distribution.

Usage

```
dmaxwell(x, rate, log = FALSE)
pmaxwell(q, rate, lower.tail = TRUE, log.p = FALSE)
qmaxwell(p, rate, lower.tail = TRUE, log.p = FALSE)
rmaxwell(n, rate)
```

Arguments

`x, q, p, n`	Same as `Uniform`.
`rate`	the (rate) parameter.
`log`	Logical. If `log = TRUE` then the logarithm of the density is returned.
`lower.tail, log.p`	Same meaning as in `pnorm` or `qnorm`.

Details

See `maxwell`, the **VGAM** family function for estimating the (rate) parameter a by maximum likelihood estimation, for the formula of the probability density function.

Value

`dmaxwell` gives the density, `pmaxwell` gives the distribution function, `qmaxwell` gives the quantile function, and `rmaxwell` generates random deviates.

Note

The Maxwell distribution is related to the Rayleigh distribution.

Author(s)

T. W. Yee and Kai Huang

References

Balakrishnan, N. and Nevzorov, V. B. (2003). *A Primer on Statistical Distributions*. Hoboken, New Jersey: Wiley.

See Also

`maxwell`, `Rayleigh`, `rayleigh`.

Examples

```
## Not run:  rate <- 3; x <- seq(-0.5, 3, length = 100)
plot(x, dmaxwell(x, rate = rate), type = "l", col = "blue",
     main = "Blue is density, orange is CDF", ylab = "", las = 1,
     sub = "Purple lines are the 10,20,...,90 percentiles")
abline(h = 0, col = "blue", lty = 2)
lines(x, pmaxwell(x, rate = rate), type = "l", col = "orange")
probs <- seq(0.1, 0.9, by = 0.1)
Q <- qmaxwell(probs, rate = rate)
lines(Q, dmaxwell(Q, rate), col = "purple", lty = 3, type = "h")
lines(Q, pmaxwell(Q, rate), col = "purple", lty = 3, type = "h")
abline(h = probs, col = "purple", lty = 3)
max(abs(pmaxwell(Q, rate) - probs))  # Should be zero

## End(Not run)
```

maxwell *Maxwell Regression Family Function*

Description

Estimating the parameter of the Maxwell distribution by maximum likelihood estimation.

Usage

```
maxwell(link = "loglink", zero = NULL, parallel = FALSE,
        type.fitted = c("mean", "percentiles", "Qlink"),
        percentiles = 50)
```

Arguments

`link`	Parameter link function applied to a, which is called the parameter `rate`. See `Links` for more choices and information; a log link is the default because the parameter is positive. More information is at `CommonVGAMffArguments`.
`zero, parallel`	See `CommonVGAMffArguments`.
`type.fitted, percentiles`	See `CommonVGAMffArguments` for information. Using `"Qlink"` is for quantile-links in **VGAMextra**.

Details

The Maxwell distribution, which is used in the area of thermodynamics, has a probability density function that can be written

$$f(y;a) = \sqrt{2/\pi} a^{3/2} y^2 \exp(-0.5ay^2)$$

for $y > 0$ and $a > 0$. The mean of Y is $\sqrt{8/(a\pi)}$ (returned as the fitted values), and its variance is $(3\pi - 8)/(\pi a)$.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, `rrvglm` and `vgam`.

Note

Fisher-scoring and Newton-Raphson are the same here. A related distribution is the Rayleigh distribution. This **VGAM** family function handles multiple responses. This **VGAM** family function can be mimicked by `poisson.points(ostatistic = 1.5, dimension = 2)`.

Author(s)

T. W. Yee

References

von Seggern, D. H. (1993). *CRC Standard Curves and Surfaces*, Boca Raton, FL, USA: CRC Press.

See Also

Maxwell, rayleigh, poisson.points.

Examples

```
mdata <- data.frame(y = rmaxwell(1000, rate = exp(2)))
fit <- vglm(y ~ 1, maxwell, mdata, trace = TRUE, crit = "coef")
coef(fit, matrix = TRUE)
Coef(fit)
```

---

mccullagh89 — *McCullagh (1989) Distribution Family Function*

Description

Estimates the two parameters of the McCullagh (1989) distribution by maximum likelihood estimation.

Usage

```
mccullagh89(ltheta = "rhobitlink", lnu = logofflink(offset = 0.5),
            itheta = NULL, inu = NULL, zero = NULL)
```

Arguments

ltheta, lnu	Link functions for the θ and ν parameters. See Links for general information.
itheta, inu	Numeric. Optional initial values for θ and ν. The default is to internally compute them.
zero	See CommonVGAMffArguments for information.

Details

The McCullagh (1989) distribution has density function

$$f(y;\theta,\nu) = \frac{\{1-y^2\}^{\nu-\frac{1}{2}}}{(1-2\theta y+\theta^2)^{\nu}\mathrm{Beta}(\nu+\frac{1}{2},\frac{1}{2})}$$

where $-1 < y < 1$ and $-1 < \theta < 1$. This distribution is equation (1) in that paper. The parameter ν satisfies $\nu > -1/2$, therefore the default is to use an log-offset link with offset equal to 0.5, i.e., $\eta_2 = \log(\nu + 0.5)$. The mean is of Y is $\nu\theta/(1+\nu)$, and these are returned as the fitted values.

This distribution is related to the Leipnik distribution (see Johnson et al. (1995)), is related to ultra-spherical functions, and under certain conditions, arises as exit distributions for Brownian motion. Fisher scoring is implemented here and it uses a diagonal matrix so the parameters are globally orthogonal in the Fisher information sense. McCullagh (1989) also states that, to some extent, θ and ν have the properties of a location parameter and a precision parameter, respectively.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, rrvglm and vgam.

Note

Convergence may be slow or fail unless the initial values are reasonably close. If a failure occurs, try assigning the argument inu and/or itheta. Figure 1 of McCullagh (1989) gives a broad range of densities for different values of θ and ν, and this could be consulted for obtaining reasonable initial values if all else fails.

Author(s)

T. W. Yee

References

McCullagh, P. (1989). Some statistical properties of a family of continuous univariate distributions. *Journal of the American Statistical Association*, **84**, 125–129.

Johnson, N. L. and Kotz, S. and Balakrishnan, N. (1995). *Continuous Univariate Distributions*, 2nd edition, Volume 2, New York: Wiley. (pages 612–617).

See Also

leipnik, rhobitlink, logofflink.

Examples

```
# Limit as theta = 0, nu = Inf:
mdata <- data.frame(y = rnorm(1000, sd = 0.2))
fit <- vglm(y ~ 1, mccullagh89, data = mdata, trace = TRUE)
head(fitted(fit))
with(mdata, mean(y))
summary(fit)
coef(fit, matrix = TRUE)
Coef(fit)
```

---

meangaitd *Mean of the GAITD Combo Density*

Description

Returns the mean of a 1- or 2-parameter GAITD combo probability mass function.

Usage

```
meangaitd(theta.p, fam = c("pois", "log", "zeta"),
    a.mix = NULL, i.mix = NULL, d.mix = NULL,
    a.mlm = NULL, i.mlm = NULL, d.mlm = NULL,
    truncate = NULL, max.support = Inf,
    pobs.mix = 0, pobs.mlm = 0,
    pstr.mix = 0, pstr.mlm = 0,
    pdip.mix = 0, pdip.mlm = 0, byrow.aid = FALSE,
    theta.a = theta.p, theta.i = theta.p, theta.d = theta.p, ...)
```

Arguments

- `theta.p` — Same as `dgaitdplot`; usually of length 1 but may be of length 2.
- `fam` — Same as `dgaitdplot`. The default is the first one. All other choices are listed in that vector.
- `a.mix, i.mix, a.mlm, i.mlm` — Same as `dgaitdplot`.
- `d.mix, d.mlm` — Same as `dgaitdplot`.
- `truncate, max.support` — Same as `dgaitdplot`.
- `pobs.mix, pobs.mlm, byrow.aid` — Same as `dgaitdplot`.
- `pstr.mix, pstr.mlm, pdip.mix, pdip.mlm` — Same as `dgaitdplot`.
- `theta.a, theta.i, theta.d` — Same as `dgaitdplot`.
- `...` — Currently unused.

Details

This function returns the mean of the PMF of the GAITD combo model. Many of its arguments are the same as `dgaitdplot`. More functionality may be added in the future, such as returning the variance.

Value

The mean.

Note

This utility function may change a lot in the future.

Author(s)

T. W. Yee.

See Also

dgaitdplot, Gaitdpois, gaitdpoisson.

Examples

```
i.mix <- seq(0, 15, by = 5)
lambda.p <- 10
meangaitd(lambda.p, a.mix = i.mix + 1, i.mix = i.mix,
          max.support = 17, pobs.mix = 0.1, pstr.mix = 0.1)
```

---

melbmaxtemp *Melbourne Daily Maximum Temperatures*

Description

Melbourne daily maximum temperatures in degrees Celsius over the ten-year period 1981–1990.

Usage

```
data(melbmaxtemp)
```

Format

A vector with 3650 observations.

Details

This is a time series data from Melbourne, Australia. It is commonly used to give a difficult quantile regression problem since the data is bimodal. That is, a hot day is likely to be followed by either an equally hot day or one much cooler. However, an independence assumption is typically made.

References

Hyndman, R. J. and Bashtannyk, D. M. and Grunwald, G. K. (1996). Estimating and visualizing conditional densities. *J. Comput. Graph. Statist.*, **5**(4), 315–336.

See Also

lms.bcn.

Examples

```
summary(melbmaxtemp)
## Not run:
melb <- data.frame(today     = melbmaxtemp[-1],
                   yesterday = melbmaxtemp[-length(melbmaxtemp)])
plot(today ~ yesterday, data = melb,
     xlab = "Yesterday's Max Temperature",
     ylab = "Today's Max Temperature", cex = 1.4, type = "n")
```

```
points(today ~ yesterday, melb, pch = 0, cex = 0.50, col = "blue")
abline(a = 0, b = 1, lty = 3)

## End(Not run)
```

---

meplot *Mean Excess Plot*

Description

Mean excess plot (also known as a mean residual life plot), a diagnostic plot for the generalized Pareto distribution (GPD).

Usage

```
meplot(object, ...)
meplot.default(y, main = "Mean Excess Plot",
    xlab = "Threshold", ylab = "Mean Excess", lty = c(2, 1:2),
    conf = 0.95, col = c("blue", "black", "blue"), type = "l", ...)
meplot.vlm(object, ...)
```

Arguments

`y`	A numerical vector. `NA`s etc. are not allowed.
`main, xlab, ylab`	Character. Overall title for the plot, and titles for the x- and y-axes.
`lty`	Line type. The second value is for the mean excess value, the first and third values are for the envelope surrounding the confidence interval.
`conf`	Confidence level. The default results in approximate 95 percent confidence intervals for each mean excess value.
`col`	Colour of the three lines.
`type`	Type of plot. The default means lines are joined between the mean excesses and also the upper and lower limits of the confidence intervals.
`object`	An object that inherits class `"vlm"`, usually of class `vglm-class` or `vgam-class`.
`...`	Graphical argument passed into `plot`. See `par` for an exhaustive list. The arguments `xlim` and `ylim` are particularly useful.

Details

If Y has a GPD with scale parameter σ and shape parameter $\xi < 1$, and if $y > 0$, then

$$E(Y - u|Y > u) = \frac{\sigma + \xi u}{1 - \xi}.$$

It is a linear function in u, the threshold. Note that $Y - u$ is called the *excess* and values of Y greater than u are called *exceedances*. The empirical versions used by these functions is to use sample means to estimate the left hand side of the equation. Values of u in the plot are the values of y itself. If the plot is roughly a straight line then the GPD is a good fit; this plot can be used to select an appropriate threshold value. See gpd for more details. If the plot is flat then the data may be exponential, and if it is curved then it may be Weibull or gamma. There is often a lot of variance/fluctuation at the RHS of the plot due to fewer observations.

The function `meplot` is generic, and `meplot.default` and `meplot.vlm` are some methods functions for mean excess plots.

Value

A list is returned invisibly with the following components.

- `threshold` The x axis values.
- `meanExcess` The y axis values. Each value is a sample mean minus a value u.
- `plusminus` The amount which is added or subtracted from the mean excess to give the confidence interval. The last value is a `NA` because it is based on one observation.

Note

The function is designed for speed and not accuracy, therefore huge data sets with extremely large values may cause failure (the function `cumsum` is used.) Ties may not be well handled.

Author(s)

T. W. Yee

References

Davison, A. C. and Smith, R. L. (1990). Models for exceedances over high thresholds (with discussion). *Journal of the Royal Statistical Society, Series B, Methodological*, **52**, 393–442.

Coles, S. (2001). *An Introduction to Statistical Modeling of Extreme Values*. London: Springer-Verlag.

See Also

gpd.

Examples

```
## Not run: meplot(with(venice90, sealevel), las = 1) -> ii
names(ii)
abline(h = ii$meanExcess[1], col = "orange", lty = "dashed")

par(mfrow = c(2, 2))
for (ii in 1:4)
  meplot(rgpd(1000), col = c("orange", "blue", "orange"))

## End(Not run)
```

micmen *Michaelis-Menten Model*

Description

Fits a Michaelis-Menten nonlinear regression model.

Usage

```
micmen(rpar = 0.001, divisor = 10, init1 = NULL, init2 = NULL,
       imethod = 1, oim = TRUE, link1 = "identitylink",
       link2 = "identitylink", firstDeriv = c("nsimEIM", "rpar"),
       probs.x = c(0.15, 0.85), nsimEIM = 500, dispersion = 0,
       zero = NULL)
```

Arguments

`rpar`	Numeric. Initial positive ridge parameter. This is used to create positive-definite weight matrices.
`divisor`	Numerical. The divisor used to divide the ridge parameter at each iteration until it is very small but still positive. The value of `divisor` should be greater than one.
`init1, init2`	Numerical. Optional initial value for the first and second parameters, respectively. The default is to use a self-starting value.
`link1, link2`	Parameter link function applied to the first and second parameters, respectively. See `Links` for more choices.
`dispersion`	Numerical. Dispersion parameter.
`firstDeriv`	Character. Algorithm for computing the first derivatives and working weights. The first is the default.
`imethod, probs.x`	See `CommonVGAMffArguments` for information.
`nsimEIM, zero`	See `CommonVGAMffArguments` for information.
`oim`	Use the OIM? See `CommonVGAMffArguments` for information.

Details

The Michaelis-Menten model is given by

$$E(Y_i) = (\theta_1 u_i)/(\theta_2 + u_i)$$

where θ_1 and θ_2 are the two parameters.

The relationship between iteratively reweighted least squares and the Gauss-Newton algorithm is given in Wedderburn (1974). However, the algorithm used by this family function is different. Details are given at the Author's web site.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Warning

This function is not (nor could ever be) entirely reliable. Plotting the fitted function and monitoring convergence is recommended.

Note

The regressor values u_i are inputted as the RHS of the form2 argument. It should just be a simple term; no smart prediction is used. It should just a single vector, therefore omit the intercept term. The LHS of the formula form2 is ignored.

To predict the response at new values of u_i one must assign the @extra$Xm2 slot in the fitted object these values, e.g., see the example below.

Numerical problems may occur. If so, try setting some initial values for the parameters. In the future, several self-starting initial values will be implemented.

Author(s)

T. W. Yee

References

Seber, G. A. F. and Wild, C. J. (1989). *Nonlinear Regression*, New York: Wiley.

Wedderburn, R. W. M. (1974). Quasi-likelihood functions, generalized linear models, and the Gauss-Newton method. *Biometrika*, **61**, 439–447.

Bates, D. M. and Watts, D. G. (1988). *Nonlinear Regression Analysis and Its Applications*, New York: Wiley.

See Also

enzyme.

Examples

```
mfit <- vglm(velocity ~ 1, micmen, data = enzyme, trace = TRUE,
             crit = "coef", form2 = ~ conc - 1)
summary(mfit)

## Not run:
plot(velocity ~ conc, enzyme, xlab = "concentration", las = 1,
     col = "blue",
     main = "Michaelis-Menten equation for the enzyme data",
     ylim = c(0, max(velocity)), xlim = c(0, max(conc)))
points(fitted(mfit) ~ conc, enzyme, col = 2, pch = "+", cex = 2)

# This predicts the response at a finer grid:
```

```
newenzyme <- data.frame(conc = seq(0, max(with(enzyme, conc)),
      len = 200))
mfit@extra$Xm2 <- newenzyme$conc # This is needed for prediction
lines(predict(mfit, newenzyme, "response") ~ conc, newenzyme,
      col = "red")
## End(Not run)
```

`mills.ratio` *Mills Ratio*

Description

Computes the Mills ratio.

Usage

```
mills.ratio(x)
mills.ratio2(x)
```

Arguments

`x`	Numeric (real).

Details

The Mills ratio here is `dnorm(x) / pnorm(x)` (some use `(1 - pnorm(x)) / dnorm(x)`). Some care is needed as `x` approaches `-Inf`; when x is very negative then its value approaches $-x$.

Value

`mills.ratio` returns the Mills ratio, and `mills.ratio2` returns `dnorm(x) * dnorm(x) / pnorm(x)`.

Author(s)

T. W. Yee

References

Mills, J. P. (1926). Table of the ratio: area to bounding ordinate, for any portion of normal curve. *Biometrika*. **18**(3/4), 395–400.

See Also

`Normal`, `tobit`, `cens.poisson`.

Examples

```
## Not run:
curve(mills.ratio, -5, 5, col = "orange", las = 1)
curve(mills.ratio, -5, 5, col = "orange", las = 1, log = "y")

## End(Not run)
```

---

mix2exp *Mixture of Two Exponential Distributions*

Description

Estimates the three parameters of a mixture of two exponential distributions by maximum likelihood estimation.

Usage

```
mix2exp(lphi = "logitlink", llambda = "loglink", iphi = 0.5,
    il1 = NULL, il2 = NULL, qmu = c(0.8, 0.2), nsimEIM = 100,
    zero = "phi")
```

Arguments

lphi, llambda	Link functions for the parameters ϕ and λ. The latter is the rate parameter and note that the mean of an ordinary exponential distribution is $1/\lambda$. See Links for more choices.
iphi, il1, il2	Initial value for ϕ, and optional initial value for λ_1 and λ_2. The last two have values that must be positive. The default is to compute initial values internally using the argument qmu.
qmu	Vector with two values giving the probabilities relating to the sample quantiles for obtaining initial values for λ_1 and λ_2. The two values are fed in as the probs argument into quantile.
nsimEIM, zero	See CommonVGAMffArguments.

Details

The probability density function can be loosely written as

$$f(y) = \phi\, Exponential(\lambda_1) + (1-\phi)\, Exponential(\lambda_2)$$

where ϕ is the probability an observation belongs to the first group, and $y > 0$. The parameter ϕ satisfies $0 < \phi < 1$. The mean of Y is $\phi/\lambda_1 + (1-\phi)/\lambda_2$ and this is returned as the fitted values. By default, the three linear/additive predictors are $(logit(\phi), \log(\lambda_1), \log(\lambda_2))^T$.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm and vgam.

Warning

This **VGAM** family function requires care for a successful application. In particular, good initial values are required because of the presence of local solutions. Therefore running this function with several different combinations of arguments such as `iphi`, `il1`, `il2`, `qmu` is highly recommended. Graphical methods such as `hist` can be used as an aid.

This **VGAM** family function is experimental and should be used with care.

Note

Fitting this model successfully to data can be difficult due to local solutions, uniqueness problems and ill-conditioned data. It pays to fit the model several times with different initial values and check that the best fit looks reasonable. Plotting the results is recommended. This function works better as λ_1 and λ_2 become more different. The default control argument `trace = TRUE` is to encourage monitoring convergence.

Author(s)

T. W. Yee

See Also

`rexp`, `exponential`, `mix2poisson`.

Examples

```
## Not run:  lambda1 <- exp(1); lambda2 <- exp(3)
(phi <- logitlink(-1, inverse = TRUE))
mdata <- data.frame(y1 = rexp(nn <- 1000, lambda1))
mdata <- transform(mdata, y2 = rexp(nn, lambda2))
mdata <- transform(mdata, Y  = ifelse(runif(nn) < phi, y1, y2))
fit <- vglm(Y ~ 1, mix2exp, data = mdata, trace = TRUE)
coef(fit, matrix = TRUE)

# Compare the results with the truth
round(rbind('Estimated' = Coef(fit),
            'Truth' = c(phi, lambda1, lambda2)), digits = 2)

with(mdata, hist(Y, prob = TRUE, main = "Orange=estimate, blue=truth"))
abline(v = 1 / Coef(fit)[c(2, 3)],  lty = 2, col = "orange", lwd = 2)
abline(v = 1 / c(lambda1, lambda2), lty = 2, col = "blue", lwd = 2)

## End(Not run)
```

`mix2normal` *Mixture of Two Univariate Normal Distributions*

Description

Estimates the five parameters of a mixture of two univariate normal distributions by maximum likelihood estimation.

Usage

```
mix2normal(lphi = "logitlink", lmu = "identitylink", lsd =
   "loglink", iphi = 0.5, imu1 = NULL, imu2 = NULL, isd1 =
   NULL, isd2 = NULL, qmu = c(0.2, 0.8), eq.sd = TRUE,
   nsimEIM = 100, zero = "phi")
```

Arguments

`lphi,lmu,lsd`	Link functions for the parameters ϕ, μ, and σ. See `Links` for more choices.
`iphi`	Initial value for ϕ, whose value must lie between 0 and 1.
`imu1, imu2`	Optional initial value for μ_1 and μ_2. The default is to compute initial values internally using the argument `qmu`.
`isd1, isd2`	Optional initial value for σ_1 and σ_2. The default is to compute initial values internally based on the argument `qmu`. Currently these are not great, therefore using these arguments where practical is a good idea.
`qmu`	Vector with two values giving the probabilities relating to the sample quantiles for obtaining initial values for μ_1 and μ_2. The two values are fed in as the `probs` argument into `quantile`.
`eq.sd`	Logical indicating whether the two standard deviations should be constrained to be equal. If `TRUE` then the appropriate constraint matrices will be used.
`nsimEIM`	See `CommonVGAMffArguments`.
`zero`	May be an integer vector specifying which linear/additive predictors are modelled as intercept-only. If given, the value or values can be from the set $\{1, 2, \ldots, 5\}$. The default is the first one only, meaning ϕ is a single parameter even when there are explanatory variables. Set `zero = NULL` to model all linear/additive predictors as functions of the explanatory variables. See `CommonVGAMffArguments` for more information.

Details

The probability density function can be loosely written as

$$f(y) = \phi\, N(\mu_1, \sigma_1) + (1 - \phi)\, N(\mu_2, \sigma_2)$$

where ϕ is the probability an observation belongs to the first group. The parameters μ_1 and μ_2 are the means, and σ_1 and σ_2 are the standard deviations. The parameter ϕ satisfies $0 < \phi < 1$.

The mean of Y is $\phi\mu_1 + (1-\phi)\mu_2$ and this is returned as the fitted values. By default, the five linear/additive predictors are $(logit(\phi), \mu_1, \log(\sigma_1), \mu_2, \log(\sigma_2))^T$. If `eq.sd = TRUE` then $\sigma_1 = \sigma_2$ is enforced.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Warning

Numerical problems can occur and half-stepping is not uncommon. If failure to converge occurs, try inputting better initial values, e.g., by using iphi, qmu, imu1, imu2, isd1, isd2, etc.

This **VGAM** family function is experimental and should be used with care.

Note

Fitting this model successfully to data can be difficult due to numerical problems and ill-conditioned data. It pays to fit the model several times with different initial values and check that the best fit looks reasonable. Plotting the results is recommended. This function works better as μ_1 and μ_2 become more different.

Convergence can be slow, especially when the two component distributions are not well separated. The default control argument `trace = TRUE` is to encourage monitoring convergence. Having `eq.sd = TRUE` often makes the overall optimization problem easier.

Author(s)

T. W. Yee

References

McLachlan, G. J. and Peel, D. (2000). *Finite Mixture Models*. New York: Wiley.

Everitt, B. S. and Hand, D. J. (1981). *Finite Mixture Distributions*. London: Chapman & Hall.

See Also

uninormal, Normal, mix2poisson.

Examples

```
## Not run:  mu1 <-  99; mu2 <- 150; nn <- 1000
sd1 <- sd2 <- exp(3)
(phi <- logitlink(-1, inverse = TRUE))
rrn <- runif(nn)
mdata <- data.frame(y = ifelse(rrn < phi, rnorm(nn, mu1, sd1),
                                          rnorm(nn, mu2, sd2)))
fit <- vglm(y ~ 1, mix2normal(eq.sd = TRUE), data = mdata)

# Compare the results
cfit <- coef(fit)
```

```
round(rbind('Estimated' = c(logitlink(cfit[1], inverse = TRUE),
            cfit[2], exp(cfit[3]), cfit[4]),
            'Truth' = c(phi, mu1, sd1, mu2)), digits = 2)

# Plot the results
xx <- with(mdata, seq(min(y), max(y), len = 200))
plot(xx, (1-phi) * dnorm(xx, mu2, sd2), type = "l", xlab = "y",
     main = "red = estimate, blue = truth",
     col = "blue", ylab = "Density")
phi.est <- logitlink(coef(fit)[1], inverse = TRUE)
sd.est <- exp(coef(fit)[3])
lines(xx, phi*dnorm(xx, mu1, sd1), col = "blue")
lines(xx, phi.est * dnorm(xx, Coef(fit)[2], sd.est), col = "red")
lines(xx, (1-phi.est)*dnorm(xx, Coef(fit)[4], sd.est), col="red")
abline(v = Coef(fit)[c(2,4)], lty = 2, col = "red")
abline(v = c(mu1, mu2), lty = 2, col = "blue")

## End(Not run)
```

`mix2poisson` *Mixture of Two Poisson Distributions*

Description

Estimates the three parameters of a mixture of two Poisson distributions by maximum likelihood estimation.

Usage

```
mix2poisson(lphi = "logitlink", llambda = "loglink",
             iphi = 0.5, il1 = NULL, il2 = NULL,
             qmu = c(0.2, 0.8), nsimEIM = 100, zero = "phi")
```

Arguments

`lphi, llambda`	Link functions for the parameter ϕ and λ. See `Links` for more choices.
`iphi`	Initial value for ϕ, whose value must lie between 0 and 1.
`il1, il2`	Optional initial value for λ_1 and λ_2. These values must be positive. The default is to compute initial values internally using the argument `qmu`.
`qmu`	Vector with two values giving the probabilities relating to the sample quantiles for obtaining initial values for λ_1 and λ_2. The two values are fed in as the `probs` argument into `quantile`.
`nsimEIM, zero`	See `CommonVGAMffArguments`.

Details

The probability function can be loosely written as

$$P(Y = y) = \phi \, Poisson(\lambda_1) + (1 - \phi) \, Poisson(\lambda_2)$$

where ϕ is the probability an observation belongs to the first group, and $y = 0, 1, 2, \ldots$. The parameter ϕ satisfies $0 < \phi < 1$. The mean of Y is $\phi\lambda_1 + (1 - \phi)\lambda_2$ and this is returned as the fitted values. By default, the three linear/additive predictors are $(logit(\phi), \log(\lambda_1), \log(\lambda_2))^T$.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm and vgam.

Warning

This **VGAM** family function requires care for a successful application. In particular, good initial values are required because of the presence of local solutions. Therefore running this function with several different combinations of arguments such as iphi, il1, il2, qmu is highly recommended. Graphical methods such as hist can be used as an aid.

With grouped data (i.e., using the weights argument) one has to use a large value of nsimEIM; see the example below.

This **VGAM** family function is experimental and should be used with care.

Note

The response must be integer-valued since dpois is invoked.

Fitting this model successfully to data can be difficult due to local solutions and ill-conditioned data. It pays to fit the model several times with different initial values, and check that the best fit looks reasonable. Plotting the results is recommended. This function works better as λ_1 and λ_2 become more different. The default control argument trace = TRUE is to encourage monitoring convergence.

Author(s)

T. W. Yee

See Also

rpois, poissonff, mix2normal.

Examples

```
## Not run:  # Example 1: simulated data
nn <- 1000
mu1 <- exp(2.5)  # Also known as lambda1
mu2 <- exp(3)
(phi <- logitlink(-0.5, inverse = TRUE))
mdata <- data.frame(y = rpois(nn, ifelse(runif(nn) < phi, mu1, mu2)))
```

```
mfit <- vglm(y ~ 1, mix2poisson, data = mdata)
coef(mfit, matrix = TRUE)

# Compare the results with the truth
round(rbind('Estimated' = Coef(mfit), 'Truth' = c(phi, mu1, mu2)), 2)

ty <- with(mdata, table(y))
plot(names(ty), ty, type = "h", main = "Orange=estimate, blue=truth",
     ylab = "Frequency", xlab = "y")
abline(v = Coef(mfit)[-1], lty = 2, col = "orange", lwd = 2)
abline(v = c(mu1, mu2), lty = 2, col = "blue", lwd = 2)

# Example 2: London Times data (Lange, 1997, p.31)
ltdata1 <- data.frame(deaths = 0:9,
                      freq = c(162,267,271, 185,111,61,27,8,3,1))
ltdata2 <- data.frame(y = with(ltdata1, rep(deaths, freq)))

# Usually this does not work well unless nsimEIM is large
Mfit <- vglm(deaths ~ 1, weight = freq, data = ltdata1,
        mix2poisson(iphi=0.3, il1=1, il2=2.5, nsimEIM=5000))

# This works better in general
Mfit = vglm(y ~ 1, mix2poisson(iphi=0.3, il1=1, il2=2.5), ltdata2)
coef(Mfit, matrix = TRUE)
Coef(Mfit)

## End(Not run)
```

MNSs *The MNSs Blood Group System*

Description

Estimates the three independent parameters of the the MNSs blood group system.

Usage

```
MNSs(link = "logitlink", imS = NULL, ims = NULL, inS = NULL)
```

Arguments

`link`	Link function applied to the three parameters. See `Links` for more choices.
`imS, ims, inS`	Optional initial value for `mS`, `ms` and `nS` respectively. A `NULL` means they are computed internally.

Details

There are three independent parameters: `m_S`, `m_s`, `n_S`, say, so that `n_s = 1 - m_S - m_s - n_S`. We let the eta vector (transposed) be `(g(m_S), g(m_s), g(n_S))` where `g` is the link function.

Value

An object of class "vglmff" (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Note

The input can be a 6-column matrix of counts, where the columns are MS, Ms, MNS, MNs, NS, Ns (in order). Alternatively, the input can be a 6-column matrix of proportions (so each row adds to 1) and the `weights` argument is used to specify the total number of counts for each row.

Author(s)

T. W. Yee

References

Elandt-Johnson, R. C. (1971). *Probability Models and Statistical Methods in Genetics*, New York: Wiley.

See Also

`AA.Aa.aa`, `AB.Ab.aB.ab`, `ABO`, `A1A2A3`.

Examples

```
# Order matters only:
y <- cbind(MS = 295, Ms = 107, MNS = 379, MNs = 322, NS = 102, Ns = 214)
fit <- vglm(y ~ 1, MNSs("logitlink", .25, .28, .08), trace = TRUE)
fit <- vglm(y ~ 1, MNSs(link = logitlink), trace = TRUE, crit = "coef")
Coef(fit)
rbind(y, sum(y)*fitted(fit))
sqrt(diag(vcov(fit)))
```

`model.framevlm` *Construct the Model Frame of a VLM Object*

Description

This function returns a `data.frame` with the variables. It is applied to an object which inherits from class "vlm" (e.g., a fitted model of class "vglm").

Usage

```
model.framevlm(object, setupsmart = TRUE, wrapupsmart = TRUE, ...)
```

Arguments

`object` a model object from the **VGAM** R package that inherits from a *vector linear model* (VLM), e.g., a model of class "vglm".

`...` further arguments such as data, na.action, subset. See model.frame for more information on these.

`setupsmart, wrapupsmart`
Logical. Arguments to determine whether to use smart prediction.

Details

Since object is an object which inherits from class "vlm" (e.g., a fitted model of class "vglm"), the method will either returned the saved model frame used when fitting the model (if any, selected by argument model = TRUE) or pass the call used when fitting on to the default method.

This code implements *smart prediction* (see smartpred).

Value

A data.frame containing the variables used in the object plus those specified in ....

References

Chambers, J. M. (1992). *Data for models.* Chapter 3 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

See Also

model.frame, model.matrixvlm, predictvglm, smartpred.

Examples

```
# Illustrates smart prediction
pneumo <- transform(pneumo, let = log(exposure.time))
fit <- vglm(cbind(normal,mild, severe) ~ poly(c(scale(let)), 2),
            multinomial, pneumo, trace = TRUE, x = FALSE)
class(fit)

check1 <- head(model.frame(fit))
check1
check2 <- model.frame(fit, data = head(pneumo))
check2
all.equal(unlist(check1), unlist(check2))  # Should be TRUE

q0 <- head(predict(fit))
q1 <- head(predict(fit, newdata = pneumo))
q2 <- predict(fit, newdata = head(pneumo))
all.equal(q0, q1)  # Should be TRUE
all.equal(q1, q2)  # Should be TRUE
```

model.matrixqrrvglm *Construct the Model Matrix of a QRR-VGLM Object*

Description

Creates a model matrix. Two types can be returned: a large one (class "vlm" or one that inherits from this such as "vglm") or a small one (such as returned if it were of class "lm").

Usage

```
model.matrixqrrvglm(object, type = c("latvar", "lm", "vlm"), ...)
```

Arguments

`object`	an object of a class "qrrvglm", i.e., a cqo object.
`type`	Type of model (or design) matrix returned. The first is the default. The value "latvar" is model matrix mainly comprising of the latent variable values (sometimes called the *site scores*). The value "lm" is the LM matrix directly corresponding to the formula argument. The value "vlm" is the big VLM model matrix *given C*.
`...`	further arguments passed to or from other methods.

Details

This function creates one of several design matrices from object. For example, this can be a small LM object or a big VLM object.

When type = "vlm" this function calls fnumat2R() to construct the big model matrix *given C*. That is, the constrained coefficients are assumed known, so that something like a large Poisson or logistic regression is set up. This is because all responses are fitted simultaneously here. The columns are labelled in the following order and with the following prefixes: "A" for the A matrix (linear in the latent variables), "D" for the D matrix (quadratic in the latent variables), "x1." for the $B1$ matrix (usually contains the intercept; see the argument noRRR in qrrvglm.control).

Value

The design matrix *after scaling* for a regression model with the specified formula and data. By *after scaling*, it is meant that it matches the output of coef(qrrvglmObject) rather than the original scaling of the fitted object.

See Also

model.matrixvlm, cqo, vcovqrrvglm.

Examples

```
## Not run:
set.seed(1); n <- 40; p <- 3; S <- 4; myrank <- 1
mydata <- rcqo(n, p, S, Rank = myrank, es.opt = TRUE, eq.max = TRUE)
(myform <- attr(mydata, "formula"))
mycqo <- cqo(myform, poissonff, data = mydata,
             I.tol = TRUE, Rank = myrank, Bestof = 5)
model.matrix(mycqo, type = "latvar")
model.matrix(mycqo, type = "lm")
model.matrix(mycqo, type = "vlm")

## End(Not run)
```

`model.matrixvlm` *Construct the Design Matrix of a VLM Object*

Description

Creates a design matrix. Two types can be returned: a large one (class "vlm" or one that inherits from this such as "vglm") or a small one (such as returned if it were of class "lm").

Usage

```
model.matrixvlm(object, type = c("vlm", "lm", "lm2", "bothlmlm2"),
                linpred.index = NULL, label.it = TRUE, ...)
```

Arguments

`object`	an object of a class that inherits from the *vector linear model* (VLM).
`type`	Type of design matrix returned. The first is the default. The value "vlm" is the VLM model matrix corresponding to the `formula` argument. The value "lm" is the LM model matrix corresponding to the `formula` argument. The value "lm2" is the second (LM) model matrix corresponding to the `form2` argument. The value "bothlmlm2" means both LM and VLM model matrices.
`linpred.index`	Vector of integers. The index for a linear/additive predictor, it must have values from the set `1:M`. Also, if `length(linpred.index) == 1` then `type = "lm"` must be assigned, whereas if `length(linpred.index) > 1` then `type = "vlm"` must be assigned, Then it returns a subset of the VLM matrix corresponding to the `linpred.index`th linear/additive predictor(s); this is a LM-type matrix when it is of unit length. Currently some attributes are returned, but these may change in value in the future because of ongoing development work.
`label.it`	Logical. Label the row and columns with character names? If `FALSE`, time and memory might be saved if the big model matrix is very large. The argument is only used when `type = "vlm"`.
`...`	further arguments passed to or from other methods. These include `data` (which is a data frame created with `model.framevlm`), `contrasts.arg`, and `xlev`. See `model.matrix` for more information.

Details

This function creates a design matrix from `object`. This can be a small LM object or a big VLM object (default). The latter is constructed from the former and the constraint matrices.

This code implements *smart prediction* (see `smartpred`).

Value

The design matrix for a regression model with the specified formula and data. If `type = "bothlmlm2"` then a list is returned with components `"X"` and `"Xm2"`.

Sometimes (especially if `x = TRUE` when calling `vglm`) the model matrix has attributes: `"assign"` (`"lm"`-type) and `"vassign"` (`"vlm"`-type) and `"orig.assign.lm"` (`"lm"`-type). These are used internally a lot for bookkeeping, especially regarding the columns of both types of model matrices. In particular, constraint matrices and variable selection relies on this information a lot. The `"orig.assign.lm"` is the ordinary `"assign"` attribute for `lm` and `glm` objects.

References

Chambers, J. M. (1992). *Data for models.* Chapter 3 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

See Also

`model.matrix`, `model.framevlm`, `predictvglm`, `smartpred`, `constraints.vlm`, `trim.constraints`, `add1.vglm`, `drop1.vglm`, `step4vglm`.

Examples

```
# (I) Illustrates smart prediction ,,,,,,,,,,,,,,,,,,,,,,,
pneumo <- transform(pneumo, let = log(exposure.time))
fit <- vglm(cbind(normal, mild, severe) ~
            sm.poly(c(sm.scale(let)), 2),
            multinomial, data = pneumo, trace = TRUE, x = FALSE)
class(fit)
fit@smart.prediction  # Data-dependent parameters
fit@x # Not saved on the object
model.matrix(fit)
model.matrix(fit, linpred.index = 1, type = "lm")
model.matrix(fit, linpred.index = 2, type = "lm")

(Check1 <- head(model.matrix(fit, type = "lm")))
(Check2 <- model.matrix(fit, data = head(pneumo), type = "lm"))
all.equal(c(Check1), c(Check2))  # Should be TRUE

q0 <- head(predict(fit))
q1 <- head(predict(fit, newdata = pneumo))
q2 <- predict(fit, newdata = head(pneumo))
all.equal(q0, q1)  # Should be TRUE
all.equal(q1, q2)  # Should be TRUE

# (II) Attributes ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
```

```
fit2 <- vglm(cbind(normal, mild, severe) ~ let,  # x = TRUE
             multinomial, data = pneumo, trace = TRUE)
fit2@x  # "lm"-type; saved on the object; note the attributes
model.matrix(fit2, type = "lm")  # Note the attributes
model.matrix(fit2, type = "vlm")  # Note the attributes
```

---

moffset *Matrix Offset*

Description

Modify a matrix by shifting successive elements.

Usage

```
moffset(mat, roffset = 0, coffset = 0, postfix = "",
        rprefix = "Row.", cprefix = "Col.")
```

Arguments

`mat` Data frame or matrix. This ought to have at least three rows and three columns. The elements are shifted in the order of `c(mat)`, i.e., going down successive columns, as the columns go from left to right. Wrapping of values is done.

`roffset, coffset` Numeric or character. If numeric, the amount of shift (offset) for each row and column. The default is no change to `mat`. If character, the offset is computed by matching with the row or column names. For example, for the `alcoff`, put `roffset = "6"` means that we make an effective day's dataset start from 6:00 am, and this wraps around to include midnight to 05.59 am on the next day.

`postfix` Character. Modified rows and columns are renamed by pasting this argument to the end of each name. The default is no change.

`rprefix, cprefix` Same as `rcim`.

Details

This function allows a matrix to be rearranged so that element (`roffset` + 1, `coffset` + 1) becomes the (1, 1) element. The elements are assumed to be ordered in the same way as the elements of `c(mat)`,

This function is applicable to, e.g., `alcoff`, where it is useful to define the *effective day* as starting at some other hour than midnight, e.g., 6.00am. This is because partying on Friday night continues on into Saturday morning, therefore it is more interpretable to use the effective day when considering a daily effect.

This is a data preprocessing function for `rcim` and `plotrcim0`. The differences between `Rcim` and `moffset` is that `Rcim` only reorders the level of the rows and columns so that the data is shifted but not moved. That is, a value in one row stays in that row, and ditto for column. But in `moffset` values in one column can be moved to a previous column. See the examples below.

Value

A matrix of the same dimensional as its input.

Note

The input `mat` should have row names and column names.

Author(s)

T. W. Yee, Alfian F. Hadi.

See Also

`Rcim`, `rcim`, `plotrcim0`, `alcoff`, `crashi`.

Examples

```
# Some day's data is moved to previous day:
moffset(alcoff, 3, 2, "*")
Rcim(alcoff, 3 + 1, 2 + 1)  # Data does not move as much.
alcoff  # Original data
moffset(alcoff, 3, 2, "*") -
Rcim(alcoff, 3+1, 2+1)  # Note the differences

# An 'effective day' data set:
alcoff.e <- moffset(alcoff, roffset = "6", postfix = "*")
fit.o <- rcim(alcoff)    # default baselines are 1st row and col
fit.e <- rcim(alcoff.e)  # default baselines are 1st row and col

## Not run:  par(mfrow = c(2, 2), mar = c(9, 4, 2, 1))
plot(fit.o, rsub = "Not very interpretable",
            csub = "Not very interpretable")
plot(fit.e, rsub = "More interpretable",
            csub = "More interpretable")

## End(Not run)

# Some checking
all.equal(moffset(alcoff), alcoff)  # Should be no change
moffset(alcoff, 1, 1, "*")
moffset(alcoff, 2, 3, "*")
moffset(alcoff, 1, 0, "*")
moffset(alcoff, 0, 1, "*")
moffset(alcoff, "6", "Mon", "*")  # This one is good

# Customise row and column baselines
fit2 <- rcim(Rcim(alcoff.e, rbaseline = "11", cbaseline = "Mon*"))
```

`multilogitlink` *Multi-logit Link Function*

Description

Computes the multilogit transformation, including its inverse and the first two derivatives.

Usage

```
multilogitlink(theta, refLevel = "(Last)", M = NULL, whitespace = FALSE,
       bvalue = NULL, inverse = FALSE, deriv = 0, all.derivs = FALSE,
       short = TRUE, tag = FALSE)
```

Arguments

`theta`	Numeric or character. See below for further details.
`refLevel, M, whitespace`	See `multinomial`.
`bvalue`	See `Links`.
`all.derivs`	Logical. This is currently experimental only.
`inverse, deriv, short, tag`	Details at `Links`.

Details

The `multilogitlink()` link function is a generalization of the `logitlink` link to M levels/classes. It forms the basis of the `multinomial` logit model. It is sometimes called the *multi-logit* link or the *multinomial logit* link; some people use *softmax* too. When its inverse function is computed it returns values which are positive and add to unity.

Value

For `multilogitlink` with `deriv = 0`, the multilogit of `theta`, i.e., `log(theta[, j]/theta[, M+1])` when `inverse = FALSE`, and if `inverse = TRUE` then `exp(theta[, j])/(1+rowSums(exp(theta)))`.

For `deriv = 1`, then the function returns *d* `eta` / *d* `theta` as a function of `theta` if `inverse = FALSE`, else if `inverse = TRUE` then it returns the reciprocal.

Here, all logarithms are natural logarithms, i.e., to base *e*.

Note

Numerical instability may occur when `theta` is close to 1 or 0 (for `multilogitlink`). One way of overcoming this is to use, e.g., `bvalue`. Currently `care.exp()` is used to avoid `NA`s being returned if the probability is too close to 1.

Author(s)

Thomas W. Yee

References

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*, 2nd ed. London: Chapman & Hall.

See Also

Links, multinomial, logitlink, gaitdpoisson, normal.vcm, CommonVGAMffArguments.

Examples

```
pneumo <- transform(pneumo, let = log(exposure.time))
fit <- vglm(cbind(normal, mild, severe) ~ let,  # For illustration only!
            multinomial, trace = TRUE, data = pneumo)
fitted(fit)
predict(fit)

multilogitlink(fitted(fit))
multilogitlink(fitted(fit)) - predict(fit)  # Should be all 0s

multilogitlink(predict(fit), inverse = TRUE)  # rowSums() add to unity
multilogitlink(predict(fit), inverse = TRUE, refLevel = 1)
multilogitlink(predict(fit), inverse = TRUE) -
fitted(fit)  # Should be all 0s

multilogitlink(fitted(fit), deriv = 1)
multilogitlink(fitted(fit), deriv = 2)
```

`multinomial` *Multinomial Logit Model*

Description

Fits a multinomial logit model (MLM) to a (preferably unordered) factor response.

Usage

```
multinomial(zero = NULL, parallel = FALSE, nointercept = NULL,
     refLevel = "(Last)", ynames = FALSE,
     imethod = 1, imu = NULL, byrow.arg = FALSE,
     Thresh = NULL, Trev = FALSE,
     Tref = if (Trev) "M" else 1, whitespace = FALSE)
```

Arguments

`zero` Can be an integer-valued vector specifying which linear/additive predictors are modelled as intercepts only. Any values must be from the set $\{1,2,\ldots,M\}$. The default value means none are modelled as intercept-only terms. See CommonVGAMffArguments for more information.

`parallel` A logical, or formula specifying which terms have equal/unequal coefficients.

`ynames` Logical. If `TRUE` then `"mu[,1]"` is replaced by the probability of the first named response category, etc. (e.g., `"P[normal]"`), so that the output is more readable, albeit less compact. This is seen in output such as `predict(fit)` and `coef(fit, matrix = TRUE)`. Of course, `"mu"` stands for the fitted probabilities, and it remains the default for upward compatibility and predictability.

`nointercept, whitespace`
See `CommonVGAMffArguments` for details.

`imu, byrow.arg` See `CommonVGAMffArguments` for details.

`refLevel` Either a (1) single positive integer or (2) a value of the factor or (3) a character string. If inputted as an integer then it specifies which column of the response matrix is the reference or baseline level. The default is the *last* one (the $(M+1)$th one). If used, this argument will be usually assigned the value `1`. If inputted as a value of a factor then beware of missing values of certain levels of the factor (`drop.unused.levels = TRUE` or `drop.unused.levels = FALSE`). See the example below. If inputted as a character string then this should be equal to (A) one of the levels of the factor response, else (B) one of the column names of the matrix response of counts; e.g., `vglm(cbind(normal, mild, severe) ~ let, multinomial(refLevel = "severe"), data = pneumo)` if it was (incorrectly because the response is ordinal) applied to the `pneumo` data set. Another example is `vglm(ethnicity ~ age, multinomial(refLevel = "European"), data = xs.nz)` if it was applied to the `xs.nz` data set.

`imethod` Choosing 2 will use the mean sample proportions of each column of the response matrix, which corresponds to the MLEs for intercept-only models. See `CommonVGAMffArguments` for more details.

`Thresh, Trev, Tref`
Same as `cumulative`. Because these arguments concern the intercepts, they should not be confused with the *stereotype* model where they would be applied to the **A** matrix instead.

Details

In this help file the response Y is assumed to be a factor with unordered values $1, 2, \dots, M+1$, so that M is the number of linear/additive predictors η_j.

The default model can be written

$$\eta_j = \log(P[Y = j]/P[Y = M+1])$$

where η_j is the jth linear/additive predictor. Here, $j = 1, \dots, M$, and η_{M+1} is 0 by definition. That is, the last level of the factor, or last column of the response matrix, is taken as the reference level or baseline—this is for identifiability of the parameters. The reference or baseline level can be changed with the `refLevel` argument.

In almost all the literature, the constraint matrices associated with this family of models are known. For example, setting `parallel = TRUE` will make all constraint matrices (including the intercept) equal to a vector of M 1's; to suppress the intercepts from being parallel then set `parallel = FALSE ~ 1`. If the constraint matrices are unknown and to be estimated, then this can be achieved by fitting the model as a reduced-rank vector generalized linear model (RR-VGLM; see `rrvglm`).

In particular, a multinomial logit model with unknown constraint matrices is known as a *stereotype* model (Anderson, 1984), and can be fitted with `rrvglm`.

The above details correspond to the ordinary MLM where all the levels are *altered* (in the terminology of GAITD regression).

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, `rrvglm` and `vgam`.

Warning

No check is made to verify that the response is nominal.

See `CommonVGAMffArguments` for more warnings.

Note

The response should be either a matrix of counts (with row sums that are all positive), or a factor. In both cases, the `y` slot returned by `vglm`/`vgam`/`rrvglm` is the matrix of sample proportions.

The multinomial logit model is more appropriate for a nominal (unordered) factor response than for an ordinal (ordered) factor response. Models more suited for the latter include those based on cumulative probabilities, e.g., `cumulative`.

`multinomial` is prone to numerical difficulties if the groups are separable and/or the fitted probabilities are close to 0 or 1. The fitted values returned are estimates of the probabilities $P[Y = j]$ for $j = 1, \ldots, M + 1$. See **safeBinaryRegression** for the logistic regression case.

Here is an example of the usage of the `parallel` argument. If there are covariates `x2`, `x3` and `x4`, then `parallel = TRUE ~ x2 + x3 - 1` and `parallel = FALSE ~ x4` are equivalent. This would constrain the regression coefficients for `x2` and `x3` to be equal; those of the intercepts and `x4` would be different.

In Example 4 below, a conditional logit model is fitted to an artificial data set that explores how cost and travel time affect people's decision about how to travel to work. Walking is the baseline group. The variable `Cost.car` is the difference between the cost of travel to work by car and walking, etc. The variable `Time.car` is the difference between the travel duration/time to work by car and walking, etc. For other details about the `xij` argument see `vglm.control` and `fill1`.

The `multinom` function in the **nnet** package uses the first level of the factor as baseline, whereas the last level of the factor is used here. Consequently the estimated regression coefficients differ.

Author(s)

Thomas W. Yee

References

Agresti, A. (2013). *Categorical Data Analysis*, 3rd ed. Hoboken, NJ, USA: Wiley.

Anderson, J. A. (1984). Regression and ordered categorical variables. *Journal of the Royal Statistical Society, Series B, Methodological*, **46**, 1–30.

Hastie, T. J., Tibshirani, R. J. and Friedman, J. H. (2009). *The Elements of Statistical Learning: Data Mining, Inference and Prediction*, 2nd ed. New York, USA: Springer-Verlag.

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*, 2nd ed. London: Chapman & Hall.

Tutz, G. (2012). *Regression for Categorical Data*, Cambridge: Cambridge University Press.

Yee, T. W. and Hastie, T. J. (2003). Reduced-rank vector generalized linear models. *Statistical Modelling*, **3**, 15–41.

Yee, T. W. (2010). The **VGAM** package for categorical data analysis. *Journal of Statistical Software*, **32**, 1–34. doi:10.18637/jss.v032.i10.

Yee, T. W. and Ma, C. (2024). Generally altered, inflated, truncated and deflated regression. *Statistical Science*, **39** (in press).

See Also

multilogitlink, margeff, cumulative, acat, cratio, sratio, CM.equid, CommonVGAMffArguments, dirichlet, dirmultinomial, rrvglm, fill1, Multinomial, gaitdpoisson, Gaitdpois, iris.

Examples

```
# Example 1: fit a MLM to Edgar Anderson's iris data
data(iris)
## Not run:  fit1 <- vglm(Species ~ ., multinomial, iris)
coef(fit1, matrix = TRUE)
fit2 <- vglm(Species ~ ., multinomial(ynames = TRUE), iris)
coef(fit2, matrix = TRUE)  # More understandable

## End(Not run)

# Example 2a: a simple example
ycounts <- t(rmultinom(10, size = 20, prob = c(0.1, 0.2, 0.8)))
fit <- vglm(ycounts ~ 1, multinomial)
head(fitted(fit))   # Proportions
fit@prior.weights   # NOT recommended for the prior weights
weights(fit, type = "prior", matrix = FALSE)  # The better method
depvar(fit)         # Sample proportions; same as fit@y
constraints(fit)    # Constraint matrices

# Example 2b: Different reference level used as the baseline
fit2 <- vglm(ycounts ~ 1, multinomial(refLevel = 2))
coef(fit2, matrix = TRUE)
coef(fit , matrix = TRUE)  # Easy to reconcile this output with fit2

# Example 3: The response is a factor.
nn <- 10
dframe3 <- data.frame(yfac = gl(3, nn, labels = c("Ctrl",
                                "Trt1", "Trt2")),
                      x2   = runif(3 * nn))
myrefLevel <- with(dframe3, yfac[12])
fit3a <- vglm(yfac ~ x2, multinomial(refLevel = myrefLevel), dframe3)
fit3b <- vglm(yfac ~ x2, multinomial(refLevel = 2), dframe3)
```

```
coef(fit3a, matrix = TRUE)  # "Trt1" is the reference level
coef(fit3b, matrix = TRUE)  # "Trt1" is the reference level
margeff(fit3b)

# Example 4: Fit a rank-1 stereotype model
fit4 <- rrvglm(Country ~ Width + Height + HP, multinomial, car.all)
coef(fit4)  # Contains the C matrix
constraints(fit4)$HP       # The A matrix
coef(fit4, matrix = TRUE)  # The B matrix
Coef(fit4)@C               # The C matrix
concoef(fit4)              # Better to get the C matrix this way
Coef(fit4)@A               # The A matrix
svd(coef(fit4, matrix = TRUE)[-1, ])$d  # Has rank 1; = C %*% t(A)
# Classification (but watch out for NAs in some of the variables):
apply(fitted(fit4), 1, which.max)  # Classification
# Classification:
colnames(fitted(fit4))[apply(fitted(fit4), 1, which.max)]
apply(predict(fit4, car.all, type = "response"),
      1, which.max)  # Ditto


# Example 5: Using the xij argument (aka conditional logit model)
set.seed(111)
nn <- 100  # Number of people who travel to work
M <- 3  # There are M+1 models of transport to go to work
ycounts <- matrix(0, nn, M+1)
ycounts[cbind(1:nn, sample(x = M+1, size = nn, replace = TRUE))] = 1
dimnames(ycounts) <- list(NULL, c("bus","train","car","walk"))
gotowork <- data.frame(cost.bus  = runif(nn), time.bus  = runif(nn),
                       cost.train= runif(nn), time.train= runif(nn),
                       cost.car  = runif(nn), time.car  = runif(nn),
                       cost.walk = runif(nn), time.walk = runif(nn))
gotowork <- round(gotowork, digits = 2)  # For convenience
gotowork <- transform(gotowork,
              Cost.bus   = cost.bus   - cost.walk,
              Cost.car   = cost.car   - cost.walk,
              Cost.train = cost.train - cost.walk,
              Cost       = cost.train - cost.walk,  # for labelling
              Time.bus   = time.bus   - time.walk,
              Time.car   = time.car   - time.walk,
              Time.train = time.train - time.walk,
              Time       = time.train - time.walk)  # for labelling
fit <- vglm(ycounts ~ Cost + Time,
            multinomial(parall = TRUE ~ Cost + Time - 1),
            xij = list(Cost ~ Cost.bus + Cost.train + Cost.car,
                       Time ~ Time.bus + Time.train + Time.car),
            form2 =  ~ Cost + Cost.bus + Cost.train + Cost.car +
                       Time + Time.bus + Time.train + Time.car,
            data = gotowork, trace = TRUE)
head(model.matrix(fit, type = "lm"))   # LM model matrix
head(model.matrix(fit, type = "vlm"))  # Big VLM model matrix
coef(fit)
coef(fit, matrix = TRUE)
```

```
constraints(fit)
summary(fit)
max(abs(predict(fit) - predict(fit, new = gotowork)))  # Should be 0
```

Nakagami *Nakagami Distribution*

Description

Density, cumulative distribution function, quantile function and random generation for the Nakagami distribution.

Usage

```
dnaka(x, scale = 1, shape, log = FALSE)
pnaka(q, scale = 1, shape, lower.tail = TRUE, log.p = FALSE)
qnaka(p, scale = 1, shape, ...)
rnaka(n, scale = 1, shape, Smallno = 1.0e-6)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. Same as in `runif`.
`scale, shape`	arguments for the parameters of the distribution. See `nakagami` for more details. For `rnaka`, arguments `shape` and `scale` must be of length 1.
`Smallno`	Numeric, a small value used by the rejection method for determining the upper limit of the distribution. That is, `pnaka(U) > 1-Smallno` where `U` is the upper limit.
`...`	Arguments that can be passed into `uniroot`.
`log`	Logical. If `log = TRUE` then the logarithm of the density is returned.
`lower.tail, log.p`	Same meaning as in `pnorm` or `qnorm`.

Details

See `nakagami` for more details.

Value

`dnaka` gives the density, `pnaka` gives the cumulative distribution function, `qnaka` gives the quantile function, and `rnaka` generates random deviates.

Author(s)

T. W. Yee and Kai Huang

See Also

nakagami.

Examples

```
## Not run:  x <- seq(0, 3.2, len = 200)
plot(x, dgamma(x, shape = 1), type = "n", col = "black", ylab = "",
     ylim = c(0,1.5), main = "dnaka(x, shape = shape)")
lines(x, dnaka(x, shape = 1), col = "orange")
lines(x, dnaka(x, shape = 2), col = "blue")
lines(x, dnaka(x, shape = 3), col = "green")
legend(2, 1.0, col = c("orange","blue","green"), lty = rep(1, len = 3),
       legend = paste("shape =", c(1, 2, 3)))

plot(x, pnorm(x), type = "n", col = "black", ylab = "",
     ylim = 0:1, main = "pnaka(x, shape = shape)")
lines(x, pnaka(x, shape = 1), col = "orange")
lines(x, pnaka(x, shape = 2), col = "blue")
lines(x, pnaka(x, shape = 3), col = "green")
legend(2, 0.6, col = c("orange","blue","green"), lty = rep(1, len = 3),
       legend = paste("shape =", c(1, 2, 3)))
## End(Not run)

probs <- seq(0.1, 0.9, by = 0.1)
pnaka(qnaka(p = probs, shape = 2), shape = 2) - probs  # Should be all 0
```

---

nakagami *Nakagami Regression Family Function*

Description

Estimation of the two parameters of the Nakagami distribution by maximum likelihood estimation.

Usage

```
nakagami(lscale = "loglink", lshape = "loglink", iscale = 1,
          ishape = NULL, nowarning = FALSE, zero = "shape")
```

Arguments

nowarning	Logical. Suppress a warning?
lscale, lshape	Parameter link functions applied to the *scale* and *shape* parameters. Log links ensure they are positive. See Links for more choices and information.
iscale, ishape	Optional initial values for the shape and scale parameters. For ishape, a NULL value means it is obtained in the initialize slot based on the value of iscale. For iscale, assigning a NULL means a value is obtained in the initialize slot, however, setting another numerical value is recommended if convergence fails or is too slow.

zero See CommonVGAMffArguments.

Details

The Nakagami distribution, which is useful for modelling wireless systems such as radio links, can be written

$$f(y) = 2(shape/scale)^{shape} y^{2 \times shape - 1} \exp(-shape \times y^2/scale)/\Gamma(shape)$$

for $y > 0$, $shape > 0$, $scale > 0$. The mean of Y is $\sqrt{scale/shape} \times \Gamma(shape + 0.5)/\Gamma(shape)$ and these are returned as the fitted values. By default, the linear/additive predictors are $\eta_1 = \log(scale)$ and $\eta_2 = \log(shape)$. Fisher scoring is implemented.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Note

The Nakagami distribution is also known as the Nakagami-*m* distribution, where $m = shape$ here. Special cases: $m = 0.5$ is a one-sided Gaussian distribution and $m = 1$ is a Rayleigh distribution. The second moment is $E(Y^2) = m$.

If Y has a Nakagami distribution with parameters *shape* and *scale* then Y^2 has a gamma distribution with shape parameter *shape* and scale parameter *scale/shape*.

Author(s)

T. W. Yee

References

Nakagami, M. (1960). The *m*-distribution: a general formula of intensity distribution of rapid fading, pp.3–36 in: *Statistical Methods in Radio Wave Propagation*. W. C. Hoffman, Ed., New York: Pergamon.

See Also

rnaka, gamma2, rayleigh.

Examples

```
nn <- 1000; shape <- exp(0); Scale <- exp(1)
ndata <- data.frame(y1 = sqrt(rgamma(nn, shape = shape, scale = Scale/shape)))
nfit <- vglm(y1 ~ 1, nakagami, data = ndata, trace = TRUE, crit = "coef")
ndata <- transform(ndata, y2 = rnaka(nn, scale = Scale, shape = shape))
nfit <- vglm(y2 ~ 1, nakagami(iscale = 3), data = ndata, trace = TRUE)
head(fitted(nfit))
with(ndata, mean(y2))
coef(nfit, matrix = TRUE)
(Cfit <- Coef(nfit))
```

```
## Not run:  sy <- with(ndata, sort(y2))
hist(with(ndata, y2), prob = TRUE, main = "", xlab = "y", ylim = c(0, 0.6),
     col = "lightblue")
lines(dnaka(sy, scale = Cfit["scale"], shape = Cfit["shape"]) ~ sy,
      data = ndata, col = "orange")
## End(Not run)
```

`nbcanlink` *Negative Binomial Canonical Link Function*

Description

Computes the negative binomial canonical link transformation, including its inverse and the first two derivatives.

Usage

```
nbcanlink(theta, size = NULL, wrt.param = NULL, bvalue = NULL,
          inverse = FALSE, deriv = 0, short = TRUE, tag = FALSE)
```

Arguments

`theta` Numeric or character. Typically the mean of a negative binomial distribution (NBD). See below for further details.

`size, wrt.param`
: `size` contains the k matrix which must be of a conformable dimension as `theta`. Also, if `deriv > 0` then `wrt.param` is either 1 or 2 (1 for with respect to the first parameter, and 2 for with respect to the second parameter (`size`)).

`bvalue` Details at `Links`.

`inverse, deriv, short, tag`
: Details at `Links`.

Details

The NBD canonical link is $\log(\theta/(\theta + k))$ where θ is the NBD mean. The canonical link is used for theoretically relating the NBD to GLM class.

This link function was specifically written for `negbinomial` and `negbinomial.size`, and should not be used elsewhere (these **VGAM** family functions have code that specifically handles `nbcanlink()`.)

Estimation with the NB canonical link has a somewhat interesting history. If we take the problem as beginning with the admission of McCullagh and Nelder (1983; first edition, p.195) [see also McCullagh and Nelder (1989, p.374)] that the NB is little used in applications and has a "problematical" canonical link then it appears only one other publicized attempt was made to solve the problem seriously. This was Hilbe, who produced a defective solution. However, Miranda and Yee (2023) solve this four-decade old problem using total derivatives and it is implemented by using `nbcanlink` with `negbinomial`. Note that early versions of **VGAM** had a defective solution.

Value

For `deriv = 0`, the above equation when `inverse = FALSE`, and if `inverse = TRUE` then `kmatrix / expm1(-theta)` where `theta` is really `eta`. For `deriv = 1`, then the function returns *d* `eta` / *d* `theta` as a function of `theta` if `inverse = FALSE`, else if `inverse = TRUE` then it returns the reciprocal.

Note

While theoretically nice, this function is not recommended in general since its value is always negative (linear predictors ought to be unbounded in general). A `loglink` link for argument `lmu` is recommended instead.

Numerical instability may occur when `theta` is close to 0 or 1. Values of `theta` which are less than or equal to 0 can be replaced by `bvalue` before computing the link function value. See `Links`.

Author(s)

Victor Miranda and Thomas W. Yee.

References

Hilbe, J. M. (2011). *Negative Binomial Regression*, 2nd Edition. Cambridge: Cambridge University Press.

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*, 2nd ed. London: Chapman & Hall.

Miranda-Soberanis, V. F. and Yee, T. W. (2023). Two-parameter link functions, with applications to negative binomial, Weibull and quantile regression. *Computational Statistics*, **38**, 1463–1485.

Yee, T. W. (2014). Reduced-rank vector generalized linear models with two linear predictors. *Computational Statistics and Data Analysis*, **71**, 889–902.

See Also

`negbinomial`, `negbinomial.size`.

Examples

```
nbcanlink("mu", short = FALSE)

mymu <- 1:10  # Test some basic operations:
kmatrix <- cbind(runif(length(mymu)))
eta1 <- nbcanlink(mymu, size = kmatrix)
ans2 <- nbcanlink(eta1, size = kmatrix, inverse = TRUE)
max(abs(ans2 - mymu))  # Should be 0

## Not run:  mymu <- seq(0.5, 10, length = 101)
kmatrix <- matrix(10, length(mymu), 1)
plot(nbcanlink(mymu, size = kmatrix) ~ mymu, las = 1,
     type = "l", col = "blue", xlab = expression({mu}))

## End(Not run)
```

```
# Estimate the parameters from some simulated data
ndata <- data.frame(x2 = runif(nn <- 500))
ndata <- transform(ndata, eta1 = -1 - 1 * x2,  # eta1 < 0
                          size1 = exp(1),
                          size2 = exp(2))
ndata <- transform(ndata,
            mu1 = nbcanlink(eta1, size = size1, inverse = TRUE),
            mu2 = nbcanlink(eta1, size = size2, inverse = TRUE))
ndata <- transform(ndata, y1 = rnbinom(nn, mu = mu1, size1),
                          y2 = rnbinom(nn, mu = mu2, size2))
summary(ndata)

nbcfit <-
  vglm(cbind(y1, y2) ~ x2,  #  crit = "c",
       negbinomial(lmu = "nbcanlink"),
       data = ndata, trace = TRUE)
coef(nbcfit, matrix = TRUE)
summary(nbcfit)
```

---

negbinomial *Negative Binomial Distribution Family Function*

Description

Maximum likelihood estimation of the two parameters of a negative binomial distribution.

Usage

```
negbinomial(zero = "size", parallel = FALSE, deviance.arg = FALSE,
            type.fitted = c("mean", "quantiles"),
            percentiles = c(25, 50, 75), vfl = FALSE,
            mds.min = 1e-3, nsimEIM = 500, cutoff.prob = 0.999,
            eps.trig = 1e-7, max.support = 4000, max.chunk.MB = 30,
            lmu = "loglink", lsize = "loglink",
            imethod = 1, imu = NULL, iprobs.y = NULL,
            gprobs.y = ppoints(6), isize = NULL,
            gsize.mux = exp(c(-30, -20, -15, -10, -6:3)))
polya(zero = "size", type.fitted = c("mean", "prob"),
      mds.min = 1e-3, nsimEIM = 500, cutoff.prob = 0.999,
      eps.trig = 1e-7, max.support = 4000, max.chunk.MB = 30,
      lprob = "logitlink", lsize = "loglink", imethod = 1, iprob = NULL,
      iprobs.y = NULL, gprobs.y = ppoints(6), isize = NULL,
      gsize.mux = exp(c(-30, -20, -15, -10, -6:3)), imunb = NULL)
polyaR(zero = "size", type.fitted = c("mean", "prob"),
       mds.min = 1e-3, nsimEIM = 500,  cutoff.prob = 0.999,
       eps.trig = 1e-7, max.support = 4000, max.chunk.MB = 30,
       lsize = "loglink", lprob = "logitlink", imethod = 1, iprob = NULL,
       iprobs.y = NULL, gprobs.y = ppoints(6), isize = NULL,
       gsize.mux = exp(c(-30, -20, -15, -10, -6:3)), imunb = NULL)
```

Arguments

`zero` Can be an integer-valued vector, and if so, then it is usually assigned -2 or 2. Specifies which of the two linear/additive predictors are modelled as an intercept only. By default, the k parameter (after `lsize` is applied) is modelled as a single unknown number that is estimated. It can be modelled as a function of the explanatory variables by setting `zero = NULL`; this has been called a NB-H model by Hilbe (2011). A negative value means that the value is recycled, so setting -2 means all k are intercept-only. See `CommonVGAMffArguments` for more information.

`lmu, lsize, lprob`
Link functions applied to the μ, k and p parameters. See `Links` for more choices. Note that the μ, k and p parameters are the `mu`, `size` and `prob` arguments of `rnbinom` respectively. Common alternatives for `lsize` are `negloglink` and `reciprocallink`, and `logloglink` (if $k > 1$).

`imu, imunb, isize, iprob`
Optional initial values for the mean and k and p. For k, if failure to converge occurs then try different values (and/or use `imethod`). For a S-column response, `isize` can be of length S. A value `NULL` means an initial value for each response is computed internally using a gridsearch based on `gsize.mux`. The last argument is ignored if used within `cqo`; see the `iKvector` argument of `qrrvglm.control` instead. In the future `isize` and `iprob` might be depreciated.

`nsimEIM` This argument is used for computing the diagonal element of the *expected information matrix* (EIM) corresponding to k based on the *simulated Fisher scoring* (SFS) algorithm. See `CommonVGAMffArguments` for more information and the notes below. SFS is one of two algorithms for computing the EIM elements (so that both algorithms may be used on a given data set). SFS is faster than the exact method when `Qmax` is large.

`cutoff.prob` Fed into the `p` argument of `qnbinom` in order to obtain an upper limit for the approximate support of the distribution, called `Qmax`, say. Similarly, the value `1-p` is fed into the `p` argument of `qnbinom` in order to obtain a lower limit for the approximate support of the distribution, called `Qmin`, say. Hence the approximate support is `Qmin:Qmax`. This argument should be a numeric and close to 1 but never exactly 1. Used to specify how many terms of the infinite series for computing the second diagonal element of the EIM are actually used. The closer this argument is to 1, the more accurate the standard errors of the regression coefficients will be. If this argument is too small, convergence will take longer.

`max.chunk.MB, max.support`
`max.support` is used to describe the eligibility of individual observations to have their EIM computed by the *exact method*. Here, we are concerned about computing the EIM wrt k. The exact method algorithm operates separately on each response variable, and it constructs a large matrix provided that the number of columns is less than `max.support`. If so, then the computations are done in chunks, so that no more than about `max.chunk.MB` megabytes of memory is used at a time (actually, it is proportional to this amount). Regarding eligibility of this algorithm, each observation must have the length of the vector, starting from the `1-cutoff.prob` quantile and finishing up at the `cutoff.prob` quantile, less than

max.support (as its approximate support). If you have abundant memory then you might try setting max.chunk.MB = Inf, but then the computations might take a very long time. Setting max.chunk.MB = 0 or max.support = 0 will force the EIM to be computed using the SFS algorithm only (this *used to be* the default method for *all* the observations). When the fitted values of the model are large and k is small, the computation of the EIM will be costly with respect to time and memory if the exact method is used. Hence the argument max.support limits the cost in terms of time. For intercept-only models max.support is multiplied by a number (such as 10) because only one inner product needs be computed. Note: max.support is an upper bound and limits the number of terms dictated by the eps.trig argument.

mds.min
: Numeric. Minimum value of the NBD mean divided by size parameter. The closer this ratio is to 0, the closer the distribution is to a Poisson. Iterations will stop when an estimate of k is so large, relative to the mean, than it is below this threshold (this is treated as a boundary of the parameter space).

vfl
: Logical. Fit the Variance–variance Factorized Loglinear (VFL) model? If TRUE then the constraint matrix rbind(0, -1) is assigned to all covariates which are not parallel. Hence parallel must be used in conjunction with this argument to specify the set of covariates used for modelling the mean. Note that the constraint matrix for the intercept should be parallel too. Some general information is at CommonVGAMffArguments.

eps.trig
: Numeric. A small positive value used in the computation of the EIMs. It focusses on the denominator of the terms of a series. Each term in the series (that is used to approximate an infinite series) has a value greater than size / sqrt(eps.trig), thus very small terms are ignored. It's a good idea to set a smaller value that will result in more accuracy, but it will require a greater computing time (when k is close to 0). And adjustment to max.support may be needed. In particular, the quantity computed by special means is $\psi'(k) - E[\psi'(Y+k)]$, which is the difference between two trigamma. functions. It is part of the calculation of the EIM with respect to the size parameter.

gsize.mux
: Similar to gsigma in CommonVGAMffArguments. However, this grid is multiplied by the initial estimates of the NBD mean parameter. That is, it is on a relative scale rather than on an absolute scale. If the counts are very large in value then convergence fail might occur; if so, then try a smaller value such as gsize.mux = exp(-40).

type.fitted, percentiles
: See CommonVGAMffArguments for more information.

deviance.arg
: Logical. If TRUE, the deviance is computed *after* convergence. It only works in the NB-2 model. It is also necessary to set criterion = "coefficients" or half.step = FALSE since one cannot use that criterion properly for the minimization within the IRLS algorithm. It should be set TRUE when used with cqo under the fast algorithm.

imethod
: An integer with value 1 or 2 etc. which specifies the initialization method for the μ parameter. If failure to converge occurs try another value and/or else specify a value for iprobs.y and/or else specify a value for isize.

parallel
: Setting parallel = TRUE is useful in order to get something similar to quasipoisson or what is known as NB-1. If parallel = TRUE then the parallelism constraint

does not apply to any intercept term. You should set zero = NULL too if parallel = TRUE to avoid a conflict. See CommonVGAMffArguments for more information. Argument vfl requires the use of parallel to fit the VFL model.

gprobs.y A vector representing a grid; passed into the probs argument of quantile when imethod = 1 to obtain an initial value for the mean of each response. Is overwritten by any value of iprobs.y.

iprobs.y Passed into the probs argument of quantile when imethod = 1 to obtain an initial value for the mean of each response. Overwrites any value of gprobs.y. This argument might be deleted in the future.

Details

The negative binomial distribution (NBD) can be motivated in several ways, e.g., as a Poisson distribution with a mean that is gamma distributed. There are several common parametrizations of the NBD. The one used by negbinomial() uses the mean μ and an *index* parameter k, both which are positive. Specifically, the density of a random variable Y is

$$f(y;\mu,k) = \binom{y+k-1}{y} \left(\frac{\mu}{\mu+k}\right)^y \left(\frac{k}{k+\mu}\right)^k$$

where $y = 0, 1, 2, \ldots$, and $\mu > 0$ and $k > 0$. Note that the *dispersion* parameter is $1/k$, so that as k approaches infinity the NBD approaches a Poisson distribution. The response has variance $Var(Y) = \mu + \mu^2/k$. When fitted, the fitted.values slot of the object contains the estimated value of the μ parameter, i.e., of the mean $E(Y)$. It is common for some to use $\alpha = 1/k$ as the ancillary or heterogeneity parameter; so common alternatives for lsize are negloglink and reciprocallink.

For polya the density is

$$f(y;p,k) = \binom{y+k-1}{y} (1-p)^y \; p^k$$

where $y = 0, 1, 2, \ldots$, and $k > 0$ and $0 < p < 1$.

Family function polyaR() is the same as polya() except the order of the two parameters are switched. The reason is that polyaR() tries to match with rnbinom closely in terms of the argument order, etc. Should the probability parameter be of primary interest, probably, users will prefer using polya() rather than polyaR(). Possibly polyaR() will be decommissioned one day.

The NBD can be coerced into the classical GLM framework with one of the parameters being of interest and the other treated as a nuisance/scale parameter (this is implemented in the **MASS** library). The **VGAM** family function negbinomial() treats both parameters on the same footing, and estimates them both by full maximum likelihood estimation.

The parameters μ and k are independent (diagonal EIM), and the confidence region for k is extremely skewed so that its standard error is often of no practical use. The parameter $1/k$ has been used as a measure of aggregation. For the NB-C the EIM is not diagonal.

These **VGAM** family functions handle *multiple* responses, so that a response matrix can be inputted. The number of columns is the number of species, say, and setting zero = -2 means that *all* species have a k equalling a (different) intercept only.

Conlisk, et al. (2007) show that fitting the NBD to presence-absence data will result in identifiability problems. However, the model is identifiable if the response values include 0, 1 and 2.

For the NB canonical link (NB-C), its estimation has a somewhat interesting history. Some details are at `nbcanlink`.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, `rrvglm` and `vgam`.

Warning

Poisson regression corresponds to k equalling infinity. If the data is Poisson or close to Poisson, numerical problems may occur. Some corrective measures are taken, e.g., k is effectively capped (relative to the mean) during estimation to some large value and a warning is issued. And setting `stepsize = 0.5` for half stepping is probably a good idea too when the data is extreme.

The NBD is a strictly unimodal distribution. Any data set that does not exhibit a mode (somewhere in the middle) makes the estimation problem difficult. Set `trace = TRUE` to monitor convergence.

These functions are fragile; the maximum likelihood estimate of the index parameter is fraught (see Lawless, 1987). Other alternatives to `negbinomial` are to fit a NB-1 or RR-NB (aka NB-P) model; see Yee (2014). Also available are the NB-C, NB-H and NB-G. Assigning values to the `isize` argument may lead to a local solution, and smaller values are preferred over large values when using this argument.

If one wants to force SFS to be used on all observations, then set `max.support = 0` or `max.chunk.MB = 0`. If one wants to force the exact method to be used for all observations, then set `max.support = Inf`. If the computer has *much* memory, then trying `max.chunk.MB = Inf` and `max.support = Inf` may provide a small speed increase. If SFS is used at all, then the working weights (`@weights`) slot of the fitted object will be a matrix; otherwise that slot will be a `0 x 0` matrix.

An alternative to the NBD is the generalized Poisson distribution, `genpoisson1`, `genpoisson2` and `genpoisson0`, since that also handles overdispersion wrt Poisson. It has one advantage in that its EIM can be computed straightforwardly.

Yet to do: write a family function which uses the methods of moments estimator for k.

Note

These 3 functions implement 2 common parameterizations of the negative binomial (NB). Some people called the NB with integer k the *Pascal* distribution, whereas if k is real then this is the *Polya* distribution. I don't. The one matching the details of `rnbinom` in terms of p and k is `polya()`.

For `polya()` the code may fail when p is close to 0 or 1. It is not yet compatible with `cqo` or `cao`.

Suppose the response is called `ymat`. For `negbinomial()` the diagonal element of the *expected information matrix* (EIM) for parameter k involves an infinite series; consequently SFS (see `nsimEIM`) is used as the backup algorithm only. SFS should be better if `max(ymat)` is large, e.g., `max(ymat) > 1000`, or if there are any outliers in `ymat`. The default algorithm involves a finite series approximation to the support `0:Inf`; the arguments `max.memory`, `min.size` and `cutoff.prob` are pertinent.

Regardless of the algorithm used, convergence problems may occur, especially when the response has large outliers or is large in magnitude. If convergence failure occurs, try using arguments (in recommended decreasing order) `max.support`, `nsimEIM`, `cutoff.prob`, `iprobs.y`, `imethod`, `isize`, `zero`, `max.chunk.MB`.

The function negbinomial can be used by the fast algorithm in cqo, however, setting eq.tolerances = TRUE and I.tolerances = FALSE is recommended.

In the first example below (Bliss and Fisher, 1953), from each of 6 McIntosh apple trees in an orchard that had been sprayed, 25 leaves were randomly selected. On each of the leaves, the number of adult female European red mites were counted.

There are two special uses of negbinomial for handling count data. Firstly, when used by rrvglm this results in a continuum of models in between and inclusive of quasi-Poisson and negative binomial regression. This is known as a reduced-rank negative binomial model *(RR-NB)*. It fits a negative binomial log-linear regression with variance function $Var(Y) = \mu + \delta_1 \mu^{\delta_2}$ where δ_1 and δ_2 are parameters to be estimated by MLE. Confidence intervals are available for δ_2, therefore it can be decided upon whether the data are quasi-Poisson or negative binomial, if any.

Secondly, the use of negbinomial with parallel = TRUE inside vglm can result in a model similar to quasipoisson. This is named the *NB-1* model. The dispersion parameter is estimated by MLE whereas glm uses the method of moments. In particular, it fits a negative binomial log-linear regression with variance function $Var(Y) = \phi_0 \mu$ where ϕ_0 is a parameter to be estimated by MLE. Confidence intervals are available for ϕ_0.

Author(s)

Thomas W. Yee, and with a lot of help by Victor Miranda to get it going with nbcanlink.

References

Bliss, C. and Fisher, R. A. (1953). Fitting the negative binomial distribution to biological data. *Biometrics* **9**, 174–200.

Conlisk, E. and Conlisk, J. and Harte, J. (2007). The impossibility of estimating a negative binomial clustering parameter from presence-absence data: A comment on He and Gaston. *The American Naturalist* **170**, 651–654.

Hilbe, J. M. (2011). *Negative Binomial Regression*, 2nd Edition. Cambridge: Cambridge University Press.

Lawless, J. F. (1987). Negative binomial and mixed Poisson regression. *The Canadian Journal of Statistics* **15**, 209–225.

Miranda-Soberanis, V. F. and Yee, T. W. (2023). Two-parameter link functions, with applications to negative binomial, Weibull and quantile regression. *Computational Statistics*, **38**, 1463–1485.

Yee, T. W. (2014). Reduced-rank vector generalized linear models with two linear predictors. *Computational Statistics and Data Analysis*, **71**, 889–902.

Yee, T. W. (2020). The **VGAM** package for negative binomial regression. *Australian & New Zealand Journal of Statistics*, **62**, 116–131.

See Also

quasipoisson, gaitdnbinomial, poissonff, zinegbinomial, negbinomial.size (e.g., NB-G), nbcanlink (NB-C), posnegbinomial, genpoisson1, genpoisson2, genpoisson0, inv.binomial, NegBinomial, rrvglm, cao, cqo, CommonVGAMffArguments, simulate.vlm, ppoints, margeff.

Examples

```
# Example 1: apple tree data (Bliss and Fisher, 1953)
appletree <- data.frame(y = 0:7, w = c(70, 38, 17, 10, 9, 3, 2, 1))
fit <- vglm(y ~ 1, negbinomial(deviance = TRUE), data = appletree,
            weights = w, crit = "coef")  # Obtain the deviance
fit <- vglm(y ~ 1, negbinomial(deviance = TRUE), data = appletree,
            weights = w, half.step = FALSE)  # Alternative method
summary(fit)
coef(fit, matrix = TRUE)
Coef(fit)  # For intercept-only models
deviance(fit)  # NB2 only; needs 'crit="coef"' & 'deviance=T' above

# Example 2: simulated data with multiple responses
## Not run:
ndata <- data.frame(x2 = runif(nn <- 200))
ndata <- transform(ndata, y1 = rnbinom(nn, exp(1), mu = exp(3+x2)),
                          y2 = rnbinom(nn, exp(0), mu = exp(2-x2)))
fit1 <- vglm(cbind(y1, y2) ~ x2, negbinomial, ndata, trace = TRUE)
coef(fit1, matrix = TRUE)

## End(Not run)

# Example 3: large counts implies SFS is used
## Not run:
ndata <- transform(ndata, y3 = rnbinom(nn, exp(1), mu = exp(10+x2)))
with(ndata, range(y3))  # Large counts
fit2 <- vglm(y3 ~ x2, negbinomial, data = ndata, trace = TRUE)
coef(fit2, matrix = TRUE)
head(weights(fit2, type = "working"))  # Non-empty; SFS was used

## End(Not run)

# Example 4: a NB-1 to estimate a NB with Var(Y)=phi0*mu
nn <- 200  # Number of observations
phi0 <- 10  # Specify this; should be greater than unity
delta0 <- 1 / (phi0 - 1)
mydata <- data.frame(x2 = runif(nn), x3 = runif(nn))
mydata <- transform(mydata, mu = exp(2 + 3 * x2 + 0 * x3))
mydata <- transform(mydata, y3 = rnbinom(nn, delta0 * mu, mu = mu))
## Not run:
plot(y3 ~ x2, data = mydata, pch = "+", col = "blue",
     main = paste("Var(Y) = ", phi0, " * mu", sep = ""), las = 1)
## End(Not run)
nb1 <- vglm(y3 ~ x2 + x3, negbinomial(parallel = TRUE, zero = NULL),
            data = mydata, trace = TRUE)
# Extracting out some quantities:
cnb1 <- coef(nb1, matrix = TRUE)
mydiff <- (cnb1["(Intercept)", "loglink(size)"] -
           cnb1["(Intercept)", "loglink(mu)"])
delta0.hat <- exp(mydiff)
(phi.hat <- 1 + 1 / delta0.hat)  # MLE of phi
summary(nb1)
```

```
# Obtain a 95 percent confidence interval for phi0:
myvec <- rbind(-1, 1, 0, 0)
(se.mydiff <- sqrt(t(myvec) %*%  vcov(nb1) %*%  myvec))
ci.mydiff <- mydiff + c(-1.96, 1.96) * c(se.mydiff)
ci.delta0 <- ci.exp.mydiff <- exp(ci.mydiff)
(ci.phi0 <- 1 + 1 / rev(ci.delta0))  # The 95% confint for phi0
Confint.nb1(nb1)  # Quick way to get it
# cf. moment estimator:
summary(glm(y3 ~ x2 + x3, quasipoisson, mydata))$disper
```

`negbinomial.size` *Negative Binomial Distribution Family Function With Known Size*

Description

Maximum likelihood estimation of the mean parameter of a negative binomial distribution with known size parameter.

Usage

```
negbinomial.size(size = Inf, lmu = "loglink", imu = NULL,
                 iprobs.y = 0.35, imethod = 1,
                 ishrinkage = 0.95, zero = NULL)
```

Arguments

`size` Numeric, positive. Same as argument `size` of `rnbinom`. If the response is a matrix then this is recycled to a matrix of the same dimension, by row (`matrix` with `byrow = TRUE`).

`lmu, imu` Same as `negbinomial`.

`iprobs.y, imethod` Same as `negbinomial`.

`zero, ishrinkage` Same as `negbinomial`. See `CommonVGAMffArguments` for information.

Details

This **VGAM** family function estimates only the mean parameter of the negative binomial distribution. See `negbinomial` for general information. Setting `size = 1` gives what might be called the NB-G (geometric model; see Hilbe (2011)). The default, `size = Inf`, corresponds to the Poisson distribution.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, `rrvglm` and `vgam`.

Note

If `lmu = "nbcanlink"` in `negbinomial.size()` then the `size` argument here should be assigned and these values are recycled.

Author(s)

Thomas W. Yee

References

Hilbe, J. M. (2011). *Negative Binomial Regression*, 2nd Edition. Cambridge: Cambridge University Press.

Yee, T. W. (2014). Reduced-rank vector generalized linear models with two linear predictors. *Computational Statistics and Data Analysis*, **71**, 889–902.

See Also

`negbinomial`, `nbcanlink` (NB-C model), `poissonff`, `rnbinom`, `simulate.vlm`.

Examples

```
# Simulated data with various multiple responses
size1 <- exp(1); size2 <- exp(2); size3 <- exp(0); size4 <- Inf
ndata <- data.frame(x2 = runif(nn <- 1000))
ndata <- transform(ndata, eta1  = -1 - 2 * x2,  # eta1 must be negative
                          size1 = size1)
ndata <- transform(ndata,
                   mu1  = nbcanlink(eta1, size = size1, inv = TRUE))
ndata <- transform(ndata,
              y1 = rnbinom(nn, mu = mu1,         size = size1),  # NB-C
              y2 = rnbinom(nn, mu = exp(2 - x2), size = size2),
              y3 = rnbinom(nn, mu = exp(3 + x2), size = size3),  # NB-G
              y4 =   rpois(nn, lambda = exp(1 + x2)))

# Also known as NB-C with size known (Hilbe, 2011)
fit1 <- vglm(y1 ~ x2, negbinomial.size(size = size1, lmu = "nbcanlink"),
             data = ndata, trace = TRUE)
coef(fit1, matrix = TRUE)
head(fit1@misc$size)  # size saved here

fit2 <- vglm(cbind(y2, y3, y4) ~ x2, data = ndata, trace = TRUE,
             negbinomial.size(size = c(size2, size3, size4)))
coef(fit2, matrix = TRUE)
head(fit2@misc$size)  # size saved here
```

normal.vcm *Univariate Normal Distribution as a Varying-Coefficient Model*

Description

Maximum likelihood estimation of all the coefficients of a LM where each of the usual regression coefficients is modelled with other explanatory variables via parameter link functions. Thus this is a basic varying-coefficient model.

Usage

```
normal.vcm(link.list = list("(Default)" = "identitylink"),
           earg.list = list("(Default)" = list()),
           lsd = "loglink", lvar = "loglink",
           esd = list(), evar = list(),
           var.arg = FALSE, imethod = 1,
           icoefficients = NULL, isd = NULL, zero = "sd",
           sd.inflation.factor = 2.5)
```

Arguments

`link.list, earg.list`
: Link functions and extra arguments applied to the coefficients of the LM, excluding the standard deviation/variance. See `CommonVGAMffArguments` for more information. The default is for an identity link to be applied to each of the regression coefficients.

`lsd, esd, lvar, evar`
: Link function and extra argument applied to the standard deviation/variance. See `CommonVGAMffArguments` for more information. Same as `uninormal`.

`icoefficients`
: Optional initial values for the coefficients. Recycled to length $M-1$ (does not include the standard deviation/variance). Try using this argument if there is a link function that is not programmed explicitly to handle range restrictions in the `initialize` slot.

`var.arg, imethod, isd`
: Same as, or similar to, `uninormal`.

`zero`
: See `CommonVGAMffArguments` for more information. The default applies to the last one, viz. the standard deviation/variance parameter.

`sd.inflation.factor`
: Numeric, should be greater than 1. The initial value of the standard deviation is multiplied by this, unless `isd` is inputted. Experience has shown that it is safer to start off with a larger value rather than a smaller one.

Details

This function allows all the usual LM regression coefficients to be modelled as functions of other explanatory variables via parameter link functions. For example, we may want some of them to be positive. Or we may want a subset of them to be positive and add to unity. So a class of such models have been named *varying-coefficient models* (VCMs).

The usual linear model is specified through argument `form2`. As with all other **VGAM** family functions, the linear/additive predictors are specified through argument `formula`.

The `multilogitlink` link allows a subset of the coefficients to be positive and add to unity. Either none or more than one call to `multilogitlink` is allowed. The last variable will be used as the baseline/reference group, and therefore excluded from the estimation.

By default, the log of the standard deviation is the last linear/additive predictor. It is recommended that this parameter be estimated as intercept-only, for numerical stability.

Technically, the Fisher information matrix is of unit-rank for all but the last parameter (the standard deviation/variance). Hence an approximation is used that pools over all the observations.

This **VGAM** family function cannot handle multiple responses. Also, this function will probably not have the full capabilities of the class of varying-coefficient models as described by Hastie and Tibshirani (1993). However, it should be able to manage some simple models, especially involving the following links: `identitylink`, `loglink`, `logofflink`, `logloglink`, `logitlink`, `probitlink`, `cauchitlink`. `clogloglink`, `rhobitlink`, `fisherzlink`.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Warning

This **VGAM** family function is fragile. One should monitor convergence, and possibly enter initial values especially when there are non-`identity`-link functions. If the initial value of the standard deviation/variance is too small then numerical problems may occur. One trick is to fit an intercept-only only model and feed its `predict()` output into argument `etastart` of a more complicated model. The use of the `zero` argument is recommended in order to keep models as simple as possible.

Note

The standard deviation/variance parameter is best modelled as intercept-only.

Yet to do: allow an argument such as `parallel` that enables many of the coefficients to be equal. Fix a bug: `Coef()` does not work for intercept-only models.

Author(s)

T. W. Yee

References

Hastie, T. and Tibshirani, R. (1993). Varying-coefficient models. *J. Roy. Statist. Soc. Ser. B*, **55**, 757–796.

See Also

`uninormal`, `lm`.

Examples

```
ndata <- data.frame(x2 = runif(nn <- 2000))
# Note that coeff1 + coeff2 + coeff5 == 1. So try "multilogitlink".
myoffset <- 10
ndata <- transform(ndata,
           coeff1 = 0.25,  # "multilogitlink"
           coeff2 = 0.25,  # "multilogitlink"
           coeff3 = exp(-0.5),  # "loglink"
# "logofflink" link:
           coeff4 = logofflink(+0.5, offset = myoffset, inverse = TRUE),
           coeff5 = 0.50,  # "multilogitlink"
           coeff6 = 1.00,  # "identitylink"
           v2 = runif(nn),
           v3 = runif(nn),
           v4 = runif(nn),
           v5 = rnorm(nn),
           v6 = rnorm(nn))
ndata <- transform(ndata,
           Coeff1 =             0.25 - 0 * x2,
           Coeff2 =             0.25 - 0 * x2,
           Coeff3 =   logitlink(-0.5  - 1 * x2, inverse = TRUE),
           Coeff4 =  logloglink( 0.5  - 1 * x2, inverse = TRUE),
           Coeff5 =             0.50 - 0 * x2,
           Coeff6 =             1.00 + 1 * x2)
ndata <- transform(ndata,
                   y1 = coeff1 * 1 +
                        coeff2 * v2 +
                        coeff3 * v3 +
                        coeff4 * v4 +
                        coeff5 * v5 +
                        coeff6 * v6 + rnorm(nn, sd = exp(0)),
                   y2 = Coeff1 * 1 +
                        Coeff2 * v2 +
                        Coeff3 * v3 +
                        Coeff4 * v4 +
                        Coeff5 * v5 +
                        Coeff6 * v6 + rnorm(nn, sd = exp(0)))

# An intercept-only model
fit1 <- vglm(y1 ~ 1,
             form2 = ~ 1 + v2 + v3 + v4 + v5 + v6,
             normal.vcm(link.list = list("(Intercept)" = "multilogitlink",
                                         "v2"          = "multilogitlink",
                                         "v3"          = "loglink",
                                         "v4"          = "logofflink",
                                         "(Default)"   = "identitylink",
                                         "v5"          = "multilogitlink"),
                        earg.list = list("(Intercept)" = list(),
                                         "v2"          = list(),
                                         "v4"          = list(offset = myoffset),
                                         "v3"          = list(),
                                         "(Default)"   = list(),
```

```
                                        "v5"         = list()),
                          zero = c(1:2, 6)),
                data = ndata, trace = TRUE)
coef(fit1, matrix = TRUE)
summary(fit1)
# This works only for intercept-only models:
multilogitlink(rbind(coef(fit1, matrix = TRUE)[1, c(1, 2)]), inverse = TRUE)

# A model with covariate x2 for the regression coefficients
fit2 <- vglm(y2 ~ 1 + x2,
             form2 = ~ 1 + v2 + v3 + v4 + v5 + v6,
             normal.vcm(link.list = list("(Intercept)" = "multilogitlink",
                                        "v2"          = "multilogitlink",
                                        "v3"          = "logitlink",
                                        "v4"          = "loglogLink",
                                        "(Default)"   = "identitylink",
                                        "v5"          = "multilogitlink"),
                          earg.list = list("(Intercept)" = list(),
                                        "v2"          = list(),
                                        "v3"          = list(),
                                        "v4"          = list(),
                                        "(Default)"   = list(),
                                        "v5"          = list()),
                          zero = c(1:2, 6)),
             data = ndata, trace = TRUE)

coef(fit2, matrix = TRUE)
summary(fit2)
```

---

nparam.vlm *Number of Parameters*

Description

Returns the number of parameters in a fitted model object.

Usage

```
         nparam(object, ...)
     nparam.vlm(object, dpar = TRUE, ...)
    nparam.vgam(object, dpar = TRUE, linear.only = FALSE, ...)
  nparam.rrvglm(object, dpar = TRUE, ...)
 nparam.qrrvglm(object, dpar = TRUE, ...)
  nparam.rrvgam(object, dpar = TRUE, ...)
```

Arguments

object	Some **VGAM** object, for example, having class vglmff-class.
...	Other possible arguments fed into the function.

dpar	Logical, include any (estimated) dispersion parameters as a parameter?
linear.only	Logical, include only the number of linear (parametric) parameters?

Details

The code was copied from the AIC() methods functions.

Value

Returns a numeric value with the corresponding number of parameters. For vgam objects, this may be real rather than integer, because the nonlinear degrees of freedom is real-valued.

Warning

This code has not been double-checked.

Author(s)

T. W. Yee.

See Also

VGLMs are described in vglm-class; VGAMs are described in vgam-class; RR-VGLMs are described in rrvglm-class; AICvlm.

Examples

```
pneumo <- transform(pneumo, let = log(exposure.time))
(fit1 <- vglm(cbind(normal, mild, severe) ~ let, propodds, data = pneumo))
coef(fit1)
coef(fit1, matrix = TRUE)
nparam(fit1)
(fit2 <- vglm(hits ~ 1, poissonff, weights = ofreq, data = V1))
coef(fit2)
coef(fit2, matrix = TRUE)
nparam(fit2)
nparam(fit2, dpar = FALSE)
```

---

olympics — *2008 and 2012 Summer Olympic Final Medal Count Data*

Description

Final medal count, by country, for the Summer 2008 and 2012 Olympic Games.

Usage

```
data(olym08)
data(olym12)
```

Format

A data frame with 87 or 85 observations on the following 6 variables.

`rank` a numeric vector, overall ranking of the countries.

`country` a factor.

`gold` a numeric vector, number of gold medals.

`silver` a numeric vector, number of silver medals.

`bronze` a numeric vector, number of bronze medals.

`totalmedal` a numeric vector, total number of medals.

Details

The events were held during (i) August 8–24, 2008, in Beijing; and (ii) 27 July–12 August, 2012, in London.

References

The official English website was/is `http://en.beijing2008.cn` and `http://www.london2012.com`. Help from Viet Hoang Quoc is gratefully acknowledged.

See Also

grc.

Examples

```
summary(olym08)
summary(olym12)
## maybe str(olym08) ; plot(olym08) ...
## Not run:  par(mfrow = c(1, 2))
myylim <- c(0, 55)
with(head(olym08, n = 8),
barplot(rbind(gold, silver, bronze),
   col = c("gold", "grey", "brown"),  # No "silver" or "bronze"!
#          "gold", "grey71", "chocolate4",
   names.arg = country, cex.names = 0.5, ylim = myylim,
   beside = TRUE, main = "2008 Summer Olympic Final Medal Count",
   ylab = "Medal count", las = 1,
   sub = "Top 8 countries; 'gold'=gold, 'grey'=silver, 'brown'=bronze"))
with(head(olym12, n = 8),
barplot(rbind(gold, silver, bronze),
   col = c("gold", "grey", "brown"),  # No "silver" or "bronze"!
   names.arg = country, cex.names = 0.5, ylim = myylim,
   beside = TRUE, main = "2012 Summer Olympic Final Medal Count",
   ylab = "Medal count", las = 1,
   sub = "Top 8 countries; 'gold'=gold, 'grey'=silver, 'brown'=bronze"))
## End(Not run)
```

`Opt` *Optimums*

Description

Generic function for the *optimums* (or optima) of a model.

Usage

```
Opt(object, ...)
```

Arguments

`object`	An object for which the computation or extraction of an optimum (or optimums) is meaningful.
`...`	Other arguments fed into the specific methods function of the model. Sometimes they are fed into the methods function for `Coef`.

Details

Different models can define an optimum in different ways. Many models have no such notion or definition.

Optimums occur in quadratic and additive ordination, e.g., CQO or CAO. For these models the optimum is the value of the latent variable where the maximum occurs, i.e., where the fitted value achieves its highest value. For quadratic ordination models there is a formula for the optimum but for additive ordination models the optimum must be searched for numerically. If it occurs on the boundary, then the optimum is undefined. At an optimum, the fitted value of the response is called the *maximum*.

Value

The value returned depends specifically on the methods function invoked.

Note

In ordination, the optimum of a species is sometimes called the *species score*.

Author(s)

Thomas W. Yee

References

Yee, T. W. (2004). A new technique for maximum-likelihood canonical Gaussian ordination. *Ecological Monographs*, **74**, 685–701.

Yee, T. W. (2006). Constrained additive ordination. *Ecology*, **87**, 203–213.

See Also

Opt.qrrvglm, Max, Tol.

Examples

```
set.seed(111)  # This leads to the global solution
hspider[,1:6] <- scale(hspider[,1:6])  # Standardized environmental vars
p1 <- cqo(cbind(Alopacce, Alopcune, Alopfabr, Arctlute, Arctperi,
                Auloalbi, Pardlugu, Pardmont, Pardnigr, Pardpull,
                Trocterr, Zoraspin) ~
          WaterCon + BareSand + FallTwig + CoveMoss + CoveHerb + ReflLux,
          family = poissonff, data = hspider, Crow1positive = FALSE)
Opt(p1)

## Not run:
clr <- (1:(ncol(depvar(p1))+1))[-7]  # Omits yellow
persp(p1, col = clr, las = 1, main = "Vertical lines at the optimums")
abline(v = Opt(p1), lty = 2, col = clr)

## End(Not run)
```

---

ordpoisson   *Ordinal Poisson Family Function*

Description

Fits a Poisson regression where the response is ordinal (the Poisson counts are grouped between known cutpoints).

Usage

```
ordpoisson(cutpoints, countdata = FALSE, NOS = NULL,
           Levels = NULL, init.mu = NULL, parallel = FALSE,
           zero = NULL, link = "loglink")
```

Arguments

cutpoints	Numeric. The cutpoints, K_l. These must be non-negative integers. Inf values may be included. See below for further details.
countdata	Logical. Is the response (LHS of formula) in count-data format? If not then the response is a matrix or vector with values 1, 2, ..., L, say, where L is the number of levels. Such input can be generated with cut with argument labels = FALSE. If countdata = TRUE then the response is expected to be in the same format as depvar(fit) where fit is a fitted model with ordpoisson as the **VGAM** family function. That is, the response is matrix of counts with L columns (if NOS = 1).

`NOS` Integer. The number of species, or more generally, the number of response random variates. This argument must be specified when `countdata = TRUE`. Usually `NOS = 1`.

`Levels` Integer vector, recycled to length `NOS` if necessary. The number of levels for each response random variate. This argument should agree with `cutpoints`. This argument must be specified when `countdata = TRUE`.

`init.mu` Numeric. Initial values for the means of the Poisson regressions. Recycled to length `NOS` if necessary. Use this argument if the default initial values fail (the default is to compute an initial value internally).

`parallel, zero, link`
See `poissonff`. See `CommonVGAMffArguments` for information.

Details

This **VGAM** family function uses maximum likelihood estimation (Fisher scoring) to fit a Poisson regression to each column of a matrix response. The data, however, is ordinal, and is obtained from known integer cutpoints. Here, $l = 1, \ldots, L$ where L ($L \geq 2$) is the number of levels. In more detail, let $Y^* = l$ if $K_{l-1} < Y \leq K_l$ where the K_l are the cutpoints. We have $K_0 = -\infty$ and $K_L = \infty$. The response for this family function corresponds to Y^* but we are really interested in the Poisson regression of Y.

If `NOS=1` then the argument `cutpoints` is a vector $(K_1, K_2, \ldots, K_L)$ where the last value (`Inf`) is optional. If `NOS>1` then the vector should have `NOS-1 Inf` values separating the cutpoints. For example, if there are `NOS=3` responses, then something like `ordpoisson(cut = c(0, 5, 10, Inf, 20, 30, Inf, 0, 10, 40, Inf))` is valid.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Warning

The input requires care as little to no checking is done. If `fit` is the fitted object, have a look at `fit@extra` and `depvar(fit)` to check.

Note

Sometimes there are no observations between two cutpoints. If so, the arguments `Levels` and `NOS` need to be specified too. See below for an example.

Author(s)

Thomas W. Yee

References

Yee, T. W. (2020). *Ordinal ordination with normalizing link functions for count data*, (in preparation).

See Also

poissonff, polf, ordered.

Examples

```
set.seed(123)  # Example 1
x2 <- runif(n <- 1000); x3 <- runif(n)
mymu <- exp(3 - 1 * x2 + 2 * x3)
y1 <- rpois(n, lambda = mymu)
cutpts <- c(-Inf, 20, 30, Inf)
fcutpts <- cutpts[is.finite(cutpts)]  # finite cutpoints
ystar <- cut(y1, breaks = cutpts, labels = FALSE)
## Not run:
plot(x2, x3, col = ystar, pch = as.character(ystar))

## End(Not run)
table(ystar) / sum(table(ystar))
fit <- vglm(ystar ~ x2 + x3, fam = ordpoisson(cutpoi = fcutpts))
head(depvar(fit))  # This can be input if countdata = TRUE
head(fitted(fit))
head(predict(fit))
coef(fit, matrix = TRUE)
fit@extra

# Example 2: multivariate and there are no obsns between some cutpoints
cutpts2 <- c(-Inf, 0, 9, 10, 20, 70, 200, 201, Inf)
fcutpts2 <- cutpts2[is.finite(cutpts2)]  # finite cutpoints
y2 <- rpois(n, lambda = mymu)   # Same model as y1
ystar2 <- cut(y2, breaks = cutpts2, labels = FALSE)
table(ystar2) / sum(table(ystar2))
fit <- vglm(cbind(ystar,ystar2) ~ x2 + x3, fam =
            ordpoisson(cutpoi = c(fcutpts,Inf,fcutpts2,Inf),
                       Levels = c(length(fcutpts)+1,length(fcutpts2)+1),
                       parallel = TRUE), trace = TRUE)
coef(fit, matrix = TRUE)
fit@extra
constraints(fit)
summary(depvar(fit))  # Some columns have all zeros
```

---

ordsup *Ordinal Superiority Measures*

Description

Ordinal superiority measures for the linear model and cumulative link models: the probability that an observation from one distribution falls above an independent observation from the other distribution, adjusted for explanatory variables in a model.

Usage

```
ordsup(object, ...)
ordsup.vglm(object, all.vars = FALSE, confint = FALSE, ...)
```

Arguments

- `object` — A `vglm` fit. Currently it must be one of: `cumulative`, `uninormal`. The links for `cumulative` must be `logitlink` or `probitlink`, and `parallel = TRUE` is also needed. For `uninormal` the mean must use `identitylink` and model the `sd` as intercept-only.
- `all.vars` — Logical. The default is to use explanatory variables which are binary, but all variables are used (except the intercept) if set to `TRUE`.
- `confint` — Logical. If `TRUE` then `confintvglm` is called to return confidence intervals for γ and Δ. By default, Wald intervals are produced, but they can be replaced by profile intervals by setting `method = "profile"`.
- `...` — Parameters that can be fed into `confintvglm`, e.g., `level = 0.95` and `method = c("wald", "profile")`.

Details

Details are given in Agresti and Kateri (2017) and this help file draws directly from this. This function returns two quantities for comparing two groups on an ordinal categorical response variable, while adjusting for other explanatory variables. They are called "ordinal superiority" measures, and the two groups can be compared without supplementary explanatory variables. Let Y_1 and Y_2 be independent random variables from groups A and B, say, for a quantitative ordinal categorical scale. Then $\Delta = P(Y_1 > Y_2) - P(Y_2 > Y_1)$ summarizes their relative size. A second quantity is $\gamma = P(Y_1 > Y_2) - 0.5 \times P(Y_2 = Y_1)$. Then $\Delta = 2 \times \gamma - 1$. whereas $\gamma = (\Delta + 1)/2$. The range of γ is $[0, 1]$, while the range of Δ is $[-1, 1]$. The examples below are based on that paper. This function is currently implemented for a very limited number of specific models.

Value

By default, a list with components `gamma` and `Delta`, where each is a vector with elements corresponding to binary explanatory variables (i.e., 0 or 1), and if no explanatory variables are binary then a `NULL` is returned. If `confint = TRUE` then the list contains 4 more components: `lower.gamma`, `upper.gamma`, `Lower.Delta`, `Upper.Delta`.

Author(s)

Thomas W. Yee

References

Agresti, A. and Kateri, M. (2017). Ordinal probability effect measures for group comparisons in multinomial cumulative link models. *Biometrics*, **73**, 214–219.

See Also

`cumulative`, `propodds`, `uninormal`.

Examples

```
## Not run:
Mental <- read.table("http://www.stat.ufl.edu/~aa/glm/data/Mental.dat",
                     header = TRUE)  # Make take a while to load in
Mental$impair <- ordered(Mental$impair)
pfit3 <- vglm(impair ~ ses + life, data = Mental,
         cumulative(link = "probitlink", reverse = FALSE, parallel = TRUE))
coef(pfit3, matrix = TRUE)
ordsup(pfit3)  # The 'ses' variable is binary

# Fit a crude LM
fit7 <- vglm(as.numeric(impair) ~ ses + life, uninormal, data = Mental)
coef(fit7, matrix = TRUE)  # 'sd' is estimated by MLE
ordsup(fit7)
ordsup(fit7, all.vars = TRUE)  # Some output may not be meaningful
ordsup(fit7, confint = TRUE, method = "profile")

## End(Not run)
```

---

oxtemp *Oxford Temperature Data*

Description

Annual maximum temperatures collected at Oxford, UK.

Usage

```
data(oxtemp)
```

Format

A data frame with 80 observations on the following 2 variables.

maxtemp Annual maximum temperatures (in degrees Fahrenheit).

year The values 1901 to 1980.

Details

The data were collected from 1901 to 1980.

Source

Unknown.

Examples

```
## Not run:  fit <- vglm(maxtemp ~ 1, gevff, data = oxtemp, trace = TRUE)
```

Paralogistic *The Paralogistic Distribution*

Description

Density, distribution function, quantile function and random generation for the paralogistic distribution with shape parameter a and scale parameter `scale`.

Usage

```
dparalogistic(x, scale = 1, shape1.a, log = FALSE)
pparalogistic(q, scale = 1, shape1.a, lower.tail = TRUE, log.p = FALSE)
qparalogistic(p, scale = 1, shape1.a, lower.tail = TRUE, log.p = FALSE)
rparalogistic(n, scale = 1, shape1.a)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. If `length(n)` > 1, the length is taken to be the number required.
`shape1.a`	shape parameter.
`scale`	scale parameter.
`log`	Logical. If `log=TRUE` then the logarithm of the density is returned.
`lower.tail, log.p`	Same meaning as in `pnorm` or `qnorm`.

Details

See `paralogistic`, which is the **VGAM** family function for estimating the parameters by maximum likelihood estimation.

Value

`dparalogistic` gives the density, `pparalogistic` gives the distribution function, `qparalogistic` gives the quantile function, and `rparalogistic` generates random deviates.

Note

The paralogistic distribution is a special case of the 4-parameter generalized beta II distribution.

Author(s)

T. W. Yee and Kai Huang

References

Kleiber, C. and Kotz, S. (2003). *Statistical Size Distributions in Economics and Actuarial Sciences*, Hoboken, NJ, USA: Wiley-Interscience.

See Also

paralogistic, genbetaII.

Examples

```
pdata <- data.frame(y = rparalogistic(n = 3000, scale = exp(1), exp(2)))
fit <- vglm(y ~ 1, paralogistic(lss = FALSE, ishape1.a = 4.1),
            data = pdata, trace = TRUE)
coef(fit, matrix = TRUE)
Coef(fit)
```

---

paralogistic *Paralogistic Distribution Family Function*

Description

Maximum likelihood estimation of the 2-parameter paralogistic distribution.

Usage

```
paralogistic(lscale = "loglink", lshape1.a = "loglink", iscale = NULL,
    ishape1.a = NULL, imethod = 1, lss = TRUE, gscale = exp(-5:5),
    gshape1.a = seq(0.75, 4, by = 0.25), probs.y = c(0.25, 0.5, 0.75),
    zero = "shape")
```

Arguments

`lss` See CommonVGAMffArguments for important information.

`lshape1.a, lscale`
: Parameter link functions applied to the (positive) parameters a and `scale`. See Links for more choices.

`iscale, ishape1.a, imethod, zero`
: See CommonVGAMffArguments for information. For `imethod = 2` a good initial value for `ishape1.a` is needed to obtain good estimates for the other parameter.

`gscale, gshape1.a`
: See CommonVGAMffArguments for information.

`probs.y` See CommonVGAMffArguments for information.

Details

The 2-parameter paralogistic distribution is the 4-parameter generalized beta II distribution with shape parameter $p = 1$ and $a = q$. It is the 3-parameter Singh-Maddala distribution with $a = q$. More details can be found in Kleiber and Kotz (2003).

The 2-parameter paralogistic has density

$$f(y) = a^2 y^{a-1} / [b^a \{1 + (y/b)^a\}^{1+a}]$$

for $a > 0$, $b > 0$, $y \geq 0$. Here, b is the scale parameter `scale`, and a is the shape parameter. The mean is

$$E(Y) = b\,\Gamma(1 + 1/a)\,\Gamma(a - 1/a)/\Gamma(a)$$

provided $a > 1$; these are returned as the fitted values. This family function handles multiple responses.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Note

See the notes in genbetaII.

Author(s)

T. W. Yee

References

Kleiber, C. and Kotz, S. (2003). *Statistical Size Distributions in Economics and Actuarial Sciences*, Hoboken, NJ, USA: Wiley-Interscience.

See Also

Paralogistic, sinmad, genbetaII, betaII, dagum, fisk, inv.lomax, lomax, inv.paralogistic.

Examples

```
pdata <- data.frame(y = rparalogistic(n = 3000, exp(1), scale = exp(1)))
fit <- vglm(y ~ 1, paralogistic(lss = FALSE), data = pdata, trace = TRUE)
fit <- vglm(y ~ 1, paralogistic(ishape1.a = 2.3, iscale = 5),
            data = pdata, trace = TRUE)
coef(fit, matrix = TRUE)
Coef(fit)
summary(fit)
```

`Pareto` *The Pareto Distribution*

Description

Density, distribution function, quantile function and random generation for the Pareto(I) distribution with parameters `scale` and `shape`.

Usage

```
dpareto(x, scale = 1, shape, log = FALSE)
ppareto(q, scale = 1, shape, lower.tail = TRUE, log.p = FALSE)
qpareto(p, scale = 1, shape, lower.tail = TRUE, log.p = FALSE)
rpareto(n, scale = 1, shape)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. Same as in `runif`.
`scale, shape`	the α and k parameters.
`log`	Logical. If `log = TRUE` then the logarithm of the density is returned.
`lower.tail, log.p`	Same meaning as in `pnorm` or `qnorm`.

Details

See `paretoff`, the **VGAM** family function for estimating the parameter k by maximum likelihood estimation, for the formula of the probability density function and the range restrictions imposed on the parameters.

Value

`dpareto` gives the density, `ppareto` gives the distribution function, `qpareto` gives the quantile function, and `rpareto` generates random deviates.

Author(s)

T. W. Yee and Kai Huang

References

Forbes, C., Evans, M., Hastings, N. and Peacock, B. (2011). *Statistical Distributions*, Hoboken, NJ, USA: John Wiley and Sons, Fourth edition.

See Also

paretoff, ParetoIV.

Examples

```
alpha <- 3; k <- exp(1); x <- seq(2.8, 8, len = 300)
## Not run:
plot(x, dpareto(x, scale = alpha, shape = k), type = "l",
     main = "Pareto density split into 10 equal areas")
abline(h = 0, col = "blue", lty = 2)
qvec <- qpareto(seq(0.1, 0.9, by = 0.1), scale = alpha, shape = k)
lines(qvec, dpareto(qvec, scale = alpha, shape = k),
      col = "purple", lty = 3, type = "h")

## End(Not run)
pvec <- seq(0.1, 0.9, by = 0.1)
qvec <- qpareto(pvec, scale = alpha, shape = k)
ppareto(qvec, scale = alpha, shape = k)
qpareto(ppareto(qvec, scale = alpha, shape = k),
        scale = alpha, shape = k) - qvec  # Should be 0
```

---

paretoff *Pareto and Truncated Pareto Distribution Family Functions*

Description

Estimates one of the parameters of the Pareto(I) distribution by maximum likelihood estimation. Also includes the upper truncated Pareto(I) distribution.

Usage

```
paretoff(scale = NULL, lshape = "loglink")
truncpareto(lower, upper, lshape = "loglink", ishape = NULL, imethod = 1)
```

Arguments

lshape	Parameter link function applied to the parameter k. See Links for more choices. A log link is the default because k is positive.
scale	Numeric. The parameter α below. If the user inputs a number then it is assumed known with this value. The default means it is estimated by maximum likelihood estimation, which means `min(y)` is used, where `y` is the response vector.
lower, upper	Numeric. Lower and upper limits for the truncated Pareto distribution. Each must be positive and of length 1. They are called α and U below.
ishape	Numeric. Optional initial value for the shape parameter. A `NULL` means a value is obtained internally. If failure to converge occurs try specifying a value, e.g., 1 or 2.
imethod	See CommonVGAMffArguments for information. If failure to converge occurs then try specifying a value for `ishape`.

Details

A random variable Y has a Pareto distribution if

$$P[Y > y] = C/y^k$$

for some positive k and C. This model is important in many applications due to the power law probability tail, especially for large values of y.

The Pareto distribution, which is used a lot in economics, has a probability density function that can be written

$$f(y; \alpha, k) = k\alpha^k / y^{k+1}$$

for $0 < \alpha < y$ and $0 < k$. The α is called the *scale* parameter, and it is either assumed *known* or else `min(y)` is used. The parameter k is called the *shape* parameter. The mean of Y is $\alpha k/(k-1)$ provided $k > 1$. Its variance is $\alpha^2 k/((k-1)^2(k-2))$ provided $k > 2$.

The upper truncated Pareto distribution has a probability density function that can be written

$$f(y) = k\alpha^k / [y^{k+1}(1 - (\alpha/U)^k)]$$

for $0 < \alpha < y < U < \infty$ and $k > 0$. Possibly, better names for k are the *index* and *tail* parameters. Here, α and U are known. The mean of Y is $k\alpha^k(U^{1-k} - \alpha^{1-k})/[(1-k)(1-(\alpha/U)^k)]$.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Warning

The usual or unbounded Pareto distribution has two parameters (called α and k here) but the family function `paretoff` estimates only k using iteratively reweighted least squares. The MLE of the α parameter lies on the boundary and is `min(y)` where `y` is the response. Consequently, using the default argument values, the standard errors are incorrect when one does a `summary` on the fitted object. If the user inputs a value for `alpha` then it is assumed known with this value and then `summary` on the fitted object should be correct. Numerical problems may occur for small k, e.g., $k < 1$.

Note

Outside of economics, the Pareto distribution is known as the Bradford distribution.

For `paretoff`, if the estimate of k is less than or equal to unity then the fitted values will be `NA`s. Also, `paretoff` fits the Pareto(I) distribution. See `paretoIV` for the more general Pareto(IV/III/II) distributions, but there is a slight change in notation: $s = k$ and $b = \alpha$.

In some applications the Pareto law is truncated by a natural upper bound on the probability tail. The upper truncated Pareto distribution has three parameters (called α, U and k here) but the family function `truncpareto()` estimates only k. With known lower and upper limits, the ML estimator of k has the usual properties of MLEs. Aban (2006) discusses other inferential details.

Author(s)

T. W. Yee

References

Forbes, C., Evans, M., Hastings, N. and Peacock, B. (2011). *Statistical Distributions*, Hoboken, NJ, USA: John Wiley and Sons, Fourth edition.

Aban, I. B., Meerschaert, M. M. and Panorska, A. K. (2006). Parameter estimation for the truncated Pareto distribution, *Journal of the American Statistical Association*, **101**(473), 270–277.

See Also

Pareto, Truncpareto, paretoIV, gpd, benini1.

Examples

```
alpha <- 2; kay <- exp(3)
pdata <- data.frame(y = rpareto(n = 1000, scale = alpha, shape = kay))
fit <- vglm(y ~ 1, paretoff, data = pdata, trace = TRUE)
fit@extra  # The estimate of alpha is here
head(fitted(fit))
with(pdata, mean(y))
coef(fit, matrix = TRUE)
summary(fit)  # Standard errors are incorrect!!

# Here, alpha is assumed known
fit2 <- vglm(y ~ 1, paretoff(scale = alpha), data = pdata, trace = TRUE)
fit2@extra  # alpha stored here
head(fitted(fit2))
coef(fit2, matrix = TRUE)
summary(fit2)  # Standard errors are okay

# Upper truncated Pareto distribution
lower <- 2; upper <- 8; kay <- exp(2)
pdata3 <- data.frame(y = rtruncpareto(n = 100, lower = lower,
                                      upper = upper, shape = kay))
fit3 <- vglm(y ~ 1, truncpareto(lower, upper), data = pdata3, trace = TRUE)
coef(fit3, matrix = TRUE)
c(fit3@misc$lower, fit3@misc$upper)
```

ParetoIV *The Pareto(IV/III/II) Distributions*

Description

Density, distribution function, quantile function and random generation for the Pareto(IV/III/II) distributions.

Usage

```
dparetoIV(x, location = 0, scale = 1, inequality = 1, shape = 1,
          log = FALSE)
pparetoIV(q, location = 0, scale = 1, inequality = 1, shape = 1,
          lower.tail = TRUE, log.p = FALSE)
qparetoIV(p, location = 0, scale = 1, inequality = 1, shape = 1,
          lower.tail = TRUE, log.p = FALSE)
rparetoIV(n, location = 0, scale = 1, inequality = 1, shape = 1)
dparetoIII(x, location = 0, scale = 1, inequality = 1, log = FALSE)
pparetoIII(q, location = 0, scale = 1, inequality = 1,
           lower.tail = TRUE, log.p = FALSE)
qparetoIII(p, location = 0, scale = 1, inequality = 1,
           lower.tail = TRUE, log.p = FALSE)
rparetoIII(n, location = 0, scale = 1, inequality = 1)
dparetoII(x, location = 0, scale = 1, shape = 1, log = FALSE)
pparetoII(q, location = 0, scale = 1, shape = 1,
          lower.tail = TRUE, log.p = FALSE)
qparetoII(p, location = 0, scale = 1, shape = 1,
          lower.tail = TRUE, log.p = FALSE)
rparetoII(n, location = 0, scale = 1, shape = 1)
dparetoI(x, scale = 1, shape = 1, log = FALSE)
pparetoI(q, scale = 1, shape = 1,
         lower.tail = TRUE, log.p = FALSE)
qparetoI(p, scale = 1, shape = 1,
         lower.tail = TRUE, log.p = FALSE)
rparetoI(n, scale = 1, shape = 1)
```

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
n	number of observations. Same as in runif.
location	the location parameter.
scale, shape, inequality	the (positive) scale, inequality and shape parameters.
log	Logical. If log = TRUE then the logarithm of the density is returned.
lower.tail, log.p	Same meaning as in pnorm or qnorm.

Details

For the formulas and other details see paretoIV.

Value

Functions beginning with the letters d give the density, p give the distribution function, q give the quantile function, and r generates random deviates.

Note

The functions [dpqr]paretoI are the same as [dpqr]pareto except for a slight change in notation: $s = k$ and $b = \alpha$; see Pareto.

Author(s)

T. W. Yee and Kai Huang

References

Brazauskas, V. (2003). Information matrix for Pareto(IV), Burr, and related distributions. *Comm. Statist. Theory and Methods* **32**, 315–325.

Arnold, B. C. (1983). *Pareto Distributions*. Fairland, Maryland: International Cooperative Publishing House.

See Also

paretoIV, Pareto.

Examples

```
## Not run:
x <- seq(-0.2, 4, by = 0.01)
loc <- 0; Scale <- 1; ineq <- 1; shape <- 1.0
plot(x, dparetoIV(x, loc, Scale, ineq, shape), type = "l",
     main = "Blue is density, orange is the CDF", col = "blue",
     sub = "Purple are 5,10,...,95 percentiles", ylim = 0:1,
     las = 1, ylab = "")
abline(h = 0, col = "blue", lty = 2)
Q <- qparetoIV(seq(0.05, 0.95,by = 0.05), loc, Scale, ineq, shape)
lines(Q, dparetoIV(Q, loc, Scale, ineq, shape), col = "purple",
      lty = 3, type = "h")
lines(x, pparetoIV(x, loc, Scale, ineq, shape), col = "orange")
abline(h = 0, lty = 2)

## End(Not run)
```

---

paretoIV *Pareto(IV/III/II) Distribution Family Functions*

Description

Estimates three of the parameters of the Pareto(IV) distribution by maximum likelihood estimation. Some special cases of this distribution are also handled.

Usage

```
paretoIV(location = 0, lscale = "loglink", linequality = "loglink",
         lshape = "loglink", iscale = 1, iinequality = 1, ishape = NULL,
         imethod = 1)
paretoIII(location = 0, lscale = "loglink", linequality = "loglink",
          iscale = NULL, iinequality = NULL)
paretoII(location = 0, lscale = "loglink", lshape = "loglink",
         iscale = NULL, ishape = NULL)
```

Arguments

`location` Location parameter, called a below. It is assumed known.

`lscale, linequality, lshape`
Parameter link functions for the scale parameter (called b below), inequality parameter (called g below), and shape parameter (called s below). See `Links` for more choices. A log link is the default for all because all these parameters are positive.

`iscale, iinequality, ishape`
Initial values for the parameters. A `NULL` value means that it is obtained internally. If convergence failure occurs, use these arguments to input some alternative initial values.

`imethod` Method of initialization for the shape parameter. Currently only values 1 and 2 are available. Try the other value if convergence failure occurs.

Details

The Pareto(IV) distribution, which is used in actuarial science, economics, finance and telecommunications, has a cumulative distribution function that can be written

$$F(y) = 1 - [1 + ((y - a)/b)^{1/g}]^{-s}$$

for $y > a$, $b > 0$, $g > 0$ and $s > 0$. The a is called the *location* parameter, b the *scale* parameter, g the *inequality* parameter, and s the *shape* parameter.

The location parameter is assumed known otherwise the Pareto(IV) distribution will not be a regular family. This assumption is not too restrictive in modelling because in typical applications this parameter is known, e.g., in insurance and reinsurance it is pre-defined by a contract and can be represented as a deductible or a retention level.

The inequality parameter is so-called because of its interpretation in the economics context. If we choose a unit shape parameter value and a zero location parameter value then the inequality parameter is the Gini index of inequality, provided $g \leq 1$.

The fitted values are currently the median, e.g., `qparetoIV` is used for `paretoIV()`.

There are a number of special cases of the Pareto(IV) distribution. These include the Pareto(I), Pareto(II), Pareto(III), and Burr family of distributions. Denoting $PIV(a, b, g, s)$ as the Pareto(IV) distribution, the Burr distribution $Burr(b, g, s)$ is $PIV(a = 0, b, 1/g, s)$, the Pareto(III) distribution $PIII(a, b, g)$ is $PIV(a, b, g, s = 1)$, the Pareto(II) distribution $PII(a, b, s)$ is $PIV(a, b, g = 1, s)$, and the Pareto(I) distribution $PI(b, s)$ is $PIV(b, b, g = 1, s)$. Thus the Burr distribution can be fitted using the `negloglink` link function and using the default `location=0` argument. The

Pareto(I) distribution can be fitted using paretoff but there is a slight change in notation: $s = k$ and $b = \alpha$.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Warning

The Pareto(IV) distribution is very general, for example, special cases include the Pareto(I), Pareto(II), Pareto(III), and Burr family of distributions. [Johnson et al. (1994) says on p.19 that fitting Type IV by ML is very difficult and rarely attempted]. Consequently, reasonably good initial values are recommended, and convergence to a local solution may occur. For this reason setting trace=TRUE is a good idea for monitoring the convergence. Large samples are ideally required to get reasonable results.

Note

The extra slot of the fitted object has a component called "location" which stores the location parameter value(s).

Author(s)

T. W. Yee

References

Johnson N. L., Kotz S., and Balakrishnan N. (1994). *Continuous Univariate Distributions, Volume 1*, 2nd ed. New York: Wiley.

Brazauskas, V. (2003). Information matrix for Pareto(IV), Burr, and related distributions. *Comm. Statist. Theory and Methods* **32**, 315–325.

Arnold, B. C. (1983). *Pareto Distributions*. Fairland, Maryland: International Cooperative Publishing House.

See Also

ParetoIV, paretoff, gpd.

Examples

```
pdata <- data.frame(y = rparetoIV(2000, scale = exp(1),
                                  ineq = exp(-0.3), shape = exp(1)))
## Not run: par(mfrow = c(2, 1))
with(pdata, hist(y)); with(pdata, hist(log(y)))
## End(Not run)
fit <- vglm(y ~ 1, paretoIV, data = pdata, trace = TRUE)
head(fitted(fit))
summary(pdata)
coef(fit, matrix = TRUE)
```

```
Coef(fit)
summary(fit)
```

Perks *The Perks Distribution*

Description

Density, cumulative distribution function, quantile function and random generation for the Perks distribution.

Usage

```
dperks(x, scale = 1, shape, log = FALSE)
pperks(q, scale = 1, shape, lower.tail = TRUE, log.p = FALSE)
qperks(p, scale = 1, shape, lower.tail = TRUE, log.p = FALSE)
rperks(n, scale = 1, shape)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. Same as in `runif`.
`log`	Logical. If `log = TRUE` then the logarithm of the density is returned.
`lower.tail, log.p`	Same meaning as in `pnorm` or `qnorm`.
`shape, scale`	positive shape and scale parameters.

Details

See `perks` for details.

Value

`dperks` gives the density, `pperks` gives the cumulative distribution function, `qperks` gives the quantile function, and `rperks` generates random deviates.

Author(s)

T. W. Yee and Kai Huang

See Also

`perks`.

Examples

```
probs <- seq(0.01, 0.99, by = 0.01)
Shape <- exp(-1.0); Scale <- exp(1);
max(abs(pperks(qperks(p = probs, Shape, Scale),
                  Shape, Scale) - probs))  # Should be 0

## Not run:  x <- seq(-0.1, 07, by = 0.01);
plot(x, dperks(x, Shape, Scale), type = "l", col = "blue", las = 1,
     main = "Blue is density, orange is cumulative distribution function",
     sub = "Purple lines are the 10,20,...,90 percentiles",
     ylab = "", ylim = 0:1)
abline(h = 0, col = "blue", lty = 2)
lines(x, pperks(x, Shape, Scale), col = "orange")
probs <- seq(0.1, 0.9, by = 0.1)
Q <- qperks(probs, Shape, Scale)
lines(Q, dperks(Q, Shape, Scale), col = "purple", lty = 3, type = "h")
pperks(Q, Shape, Scale) - probs  # Should be all zero
abline(h = probs, col = "purple", lty = 3)
## End(Not run)
```

`perks` *Perks Distribution Family Function*

Description

Maximum likelihood estimation of the 2-parameter Perks distribution.

Usage

```
perks(lscale = "loglink", lshape = "loglink",
      iscale = NULL,   ishape = NULL,
      gscale = exp(-5:5), gshape = exp(-5:5),
      nsimEIM = 500, oim.mean = FALSE, zero = NULL,
      nowarning = FALSE)
```

Arguments

`nowarning`	Logical. Suppress a warning? Ignored for **VGAM** 0.9-7 and higher.
`lscale, lshape`	Parameter link functions applied to the shape parameter `shape`, scale parameter `scale`. All parameters are treated as positive here See `Links` for more choices.
`iscale, ishape`	Optional initial values. A `NULL` means a value is computed internally.
`gscale, gshape`	See `CommonVGAMffArguments`.
`nsimEIM, zero`	See `CommonVGAMffArguments`.
`oim.mean`	To be currently ignored.

Details

The Perks distribution has cumulative distribution function

$$F(y;\alpha,\beta) = 1 - \left\{ \frac{1+\alpha}{1+\alpha e^{\beta y}} \right\}^{1/\beta}$$

which leads to a probability density function

$$f(y;\alpha,\beta) = [1+\alpha]^{1/\beta} \, \alpha e^{\beta y} / (1+\alpha e^{\beta y})^{1+1/\beta}$$

for $\alpha > 0$, $\beta > 0$, $y > 0$. Here, β is called the scale parameter `scale`, and α is called a shape parameter. The moments for this distribution do not appear to be available in closed form.

Simulated Fisher scoring is used and multiple responses are handled.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Warning

A lot of care is needed because this is a rather difficult distribution for parameter estimation. If the self-starting initial values fail then try experimenting with the initial value arguments, especially `iscale`. Successful convergence depends on having very good initial values. Also, monitor convergence by setting `trace = TRUE`.

Author(s)

T. W. Yee

References

Perks, W. (1932). On some experiments in the graduation of mortality statistics. *Journal of the Institute of Actuaries*, **63**, 12–40.

Richards, S. J. (2012). A handbook of parametric survival models for actuarial use. *Scandinavian Actuarial Journal*. 1–25.

See Also

`dperks`, `simulate.vlm`.

Examples

```
## Not run:  set.seed(123)
pdata <- data.frame(x2 = runif(nn <- 1000))  # x2 unused
pdata <- transform(pdata, eta1  = -1,
                          ceta1 =  1)
pdata <- transform(pdata, shape1 = exp(eta1),
                          scale1 = exp(ceta1))
pdata <- transform(pdata, y1 = rperks(nn, sh = shape1, sc = scale1))
```

```
fit1 <- vglm(y1 ~ 1, perks, data = pdata, trace = TRUE)
coef(fit1, matrix = TRUE)
summary(fit1)

## End(Not run)
```

perspqrrvglm *Perspective plot for QRR-VGLMs*

Description

Produces a perspective plot for a CQO model (QRR-VGLM). It is only applicable for rank-1 or rank-2 models with argument noRRR = ~ 1.

Usage

```
perspqrrvglm(x, varI.latvar = FALSE, refResponse = NULL, show.plot = TRUE,
             xlim = NULL, ylim = NULL, zlim = NULL,
             gridlength = if (Rank == 1) 301 else c(51,51),
             which.species = NULL,
             xlab = if (Rank == 1) "Latent Variable" else "Latent Variable 1",
             ylab = if (Rank == 1) "Expected Value" else "Latent Variable 2",
             zlab = "Expected value", labelSpecies = FALSE,
             stretch = 1.05, main = "", ticktype = "detailed",
             col = if (Rank == 1) par()$col else "white",
             llty = par()$lty, llwd = par()$lwd,
             add1 = FALSE, ...)
```

Arguments

`x`	Object of class "qrrvglm", i.e., a constrained quadratic ordination (CQO) object.
`varI.latvar`	Logical that is fed into `Coef.qrrvglm`.
`refResponse`	Integer or character that is fed into `Coef.qrrvglm`.
`show.plot`	Logical. Plot it?
`xlim, ylim`	Limits of the x- and y-axis. Both are numeric of length 2. See `par`.
`zlim`	Limits of the z-axis. Numeric of length 2. Ignored if rank is 1. See `par`.
`gridlength`	Numeric. The fitted values are evaluated on a grid, and this argument regulates the fineness of the grid. If Rank = 2 then the argument is recycled to length 2, and the two numbers are the number of grid points on the x- and y-axes respectively.
`which.species`	Numeric or character vector. Indicates which species are to be plotted. The default is to plot all of them. If numeric, it should contain values in the set $\{1,2,\ldots,S\}$ where S is the number of species.
`xlab, ylab`	Character caption for the x-axis and y-axis. By default, a suitable caption is found. See the `xlab` argument in `plot` or `title`.

zlab	Character caption for the z-axis. Used only if Rank = 2. By default, a suitable caption is found. See the xlab argument in plot or title.
labelSpecies	Logical. Whether the species should be labelled with their names. Used for Rank = 1 only. The position of the label is just above the species' maximum.
stretch	Numeric. A value slightly more than 1, this argument adjusts the height of the y-axis. Used for Rank = 1 only.
main	Character, giving the title of the plot. See the main argument in plot or title.
ticktype	Tick type. Used only if Rank = 2. See persp for more information.
col	Color. See persp for more information.
llty	Line type. Rank-1 models only. See the lty argument of par.
llwd	Line width. Rank-1 models only. See the lwd argument of par.
add1	Logical. Add to an existing plot? Used only for rank-1 models.
...	Arguments passed into persp. Useful arguments here include theta and phi, which control the position of the eye.

Details

For a rank-1 model, a perspective plot is similar to lvplot.qrrvglm but plots the curves along a fine grid and there is no rugplot to show the site scores.

For a rank-2 model, a perspective plot has the first latent variable as the x-axis, the second latent variable as the y-axis, and the expected value (fitted value) as the z-axis. The result of a CQO is that each species has a response surface with elliptical contours. This function will, at each grid point, work out the maximum fitted value over all the species. The resulting response surface is plotted. Thus rare species will be obscured and abundant species will dominate the plot. To view rare species, use the which.species argument to select a subset of the species.

A perspective plot will be performed if noRRR = ~ 1, and Rank = 1 or 2. Also, all the tolerance matrices of those species to be plotted must be positive-definite.

Value

For a rank-2 model, a list with the following components.

fitted	A $(G_1 \times G_2)$ by M matrix of fitted values on the grid. Here, G_1 and G_2 are the two values of gridlength.
latvar1grid, latvar2grid	The grid points for the x-axis and y-axis.
max.fitted	A G_1 by G_2 matrix of maximum of the fitted values over all species. These are the values that are plotted on the z-axis.

For a rank-1 model, the components latvar2grid and max.fitted are NULL.

Note

Yee (2004) does not refer to perspective plots. Instead, contour plots via lvplot.qrrvglm are used.

For rank-1 models, a similar function to this one is lvplot.qrrvglm. It plots the fitted values at the actual site score values rather than on a fine grid here. The result has the advantage that the user sees the curves as a direct result from a model fitted to data whereas here, it is easy to think that the smooth bell-shaped curves are the truth because the data is more of a distance away.

Author(s)

Thomas W. Yee

References

Yee, T. W. (2004). A new technique for maximum-likelihood canonical Gaussian ordination. *Ecological Monographs*, **74**, 685–701.

See Also

persp, cqo, Coef.qrrvglm, lvplot.qrrvglm, par, title.

Examples

```
## Not run:
hspider[, 1:6] <- scale(hspider[, 1:6])  # Good idea when I.tolerances = TRUE
set.seed(111)
r1 <- cqo(cbind(Alopacce, Alopcune, Alopfabr, Arctlute, Arctperi,
                Auloalbi, Pardmont, Pardnigr, Pardpull, Trocterr) ~
          WaterCon + BareSand + FallTwig + CoveMoss + CoveHerb + ReflLux,
          poissonff, data = hspider, trace = FALSE, I.tolerances = TRUE)
set.seed(111)  # r2 below is an ill-conditioned model
r2 <- cqo(cbind(Alopacce, Alopcune, Alopfabr, Arctlute, Arctperi,
                Auloalbi, Pardmont, Pardnigr, Pardpull, Trocterr) ~
          WaterCon + BareSand + FallTwig + CoveMoss + CoveHerb + ReflLux,
          isd.lv = c(2.4, 1.0), Muxfactor = 3.0, trace = FALSE,
          poissonff, data = hspider, Rank = 2, eq.tolerances = TRUE)

sort(deviance(r1, history = TRUE))  # A history of all the fits
sort(deviance(r2, history = TRUE))  # A history of all the fits
if (deviance(r2) > 857) stop("suboptimal fit obtained")

persp(r1, xlim = c(-6, 5), col = 1:4, label = TRUE)

# Involves all species
persp(r2, xlim = c(-6, 5), ylim = c(-4, 5), theta = 10, phi = 20, zlim = c(0, 220))
# Omit the two dominant species to see what is behind them
persp(r2, xlim = c(-6, 5), ylim = c(-4, 5), theta = 10, phi = 20, zlim = c(0, 220),
      which = (1:10)[-c(8, 10)])  # Use zlim to retain the original z-scale

## End(Not run)
```

---

pgamma.deriv *Derivatives of the Incomplete Gamma Integral*

Description

The first two derivatives of the incomplete gamma integral.

Usage

```
pgamma.deriv(q, shape, tmax = 100)
```

Arguments

q, shape	As in pgamma but these must be vectors of positive values only and finite.
tmax	Maximum number of iterations allowed in the computation (per q value).

Details

Write $x = q$ and shape = a. The first and second derivatives with respect to q and a are returned. This function is similar in spirit to pgamma; define

$$P(a,x) = \frac{1}{\Gamma(a)} \int_0^x t^{a-1} e^{-t} dt$$

so that $P(a,x)$ is pgamma(x, a). Currently a 6-column matrix is returned (in the future this may change and an argument may be supplied so that only what is required by the user is computed.)

The computations use a series expansion for $a \leq x \leq 1$ or or $x < a$, else otherwise a continued fraction expansion. Machine overflow can occur for large values of x when x is much greater than a.

Value

The first 5 columns, running from left to right, are the derivatives with respect to: x, x^2, a, a^2, xa. The 6th column is $P(a,x)$ (but it is not as accurate as calling pgamma directly).

Note

If convergence does not occur then try increasing the value of tmax.

Yet to do: add more arguments to give greater flexibility in the accuracy desired and to compute only quantities that are required by the user.

Author(s)

T. W. Yee wrote the wrapper function to the Fortran subroutine written by R. J. Moore. The subroutine was modified to run using double precision. The original code came from http://lib.stat.cmu.edu/apstat/187. but this website has since become stale.

References

Moore, R. J. (1982). Algorithm AS 187: Derivatives of the Incomplete Gamma Integral. *Journal of the Royal Statistical Society, Series C (Applied Statistics)*, **31**(3), 330–335.

See Also

pgamma.deriv.unscaled, pgamma.

Examples

```
x <- seq(2, 10, length = 501)
head(ans <- pgamma.deriv(x, 2))
## Not run:  par(mfrow = c(2, 3))
for (jay in 1:6)
  plot(x, ans[, jay], type = "l", col = "blue", cex.lab = 1.5,
       cex.axis = 1.5, las = 1, log = "x",
       main = colnames(ans)[jay], xlab = "q", ylab = "")
## End(Not run)
```

`pgamma.deriv.unscaled` *Derivatives of the Incomplete Gamma Integral (Unscaled Version)*

Description

The first two derivatives of the incomplete gamma integral with scaling.

Usage

```
pgamma.deriv.unscaled(q, shape)
```

Arguments

`q, shape`	As in `pgamma` and `pgamma.deriv` but these must be vectors of positive values only and finite.

Details

Define

$$G(x,a) = \int_0^x t^{a-1} e^{-t} dt$$

so that $G(x,a)$ is `pgamma(x, a) * gamma(a)`. Write $x = q$ and `shape` = a. The 0th and first and second derivatives with respect to a of G are returned. This function is similar in spirit to `pgamma.deriv` but here there is no gamma function to scale things. Currently a 3-column matrix is returned (in the future this may change and an argument may be supplied so that only what is required by the user is computed.) This function is based on Wingo (1989).

Value

The 3 columns, running from left to right, are the `0:2`th derivatives with respect to a.

Warning

These function seems inaccurate for `q = 1` and `q = 2`; see the plot below.

Author(s)

T. W. Yee.

References

See truncweibull.

See Also

pgamma.deriv, pgamma.

Examples

```
x <- 3; aa <- seq(0.3, 04, by = 0.01)
ans.u <- pgamma.deriv.unscaled(x, aa)
head(ans.u)

## Not run:  par(mfrow = c(1, 3))
for (jay in 1:3) {
  plot(aa, ans.u[, jay], type = "l", col = "blue", cex.lab = 1.5,
       cex.axis = 1.5, las = 1, main = colnames(ans.u)[jay],
       log = "", xlab = "shape", ylab = "")
  abline(h = 0, v = 1:2, lty = "dashed", col = "gray")  # Inaccurate at 1 and 2
}

## End(Not run)
```

---

plotdeplot.lmscreg *Density Plot for LMS Quantile Regression*

Description

Plots a probability density function associated with a LMS quantile regression.

Usage

```
plotdeplot.lmscreg(answer, y.arg, add.arg = FALSE,
    xlab = "", ylab = "density", xlim = NULL, ylim = NULL,
    llty.arg = par()$lty, col.arg = par()$col,
    llwd.arg = par()$lwd, ...)
```

Arguments

answer	Output from functions of the form deplot.??? where ??? is the name of the **VGAM** LMS family function, e.g., lms.yjn. See below for details.
y.arg	Numerical vector. The values of the response variable at which to evaluate the density. This should be a grid that is fine enough to ensure the plotted curves are smooth.
add.arg	Logical. Add the density to an existing plot?
xlab, ylab	Caption for the x- and y-axes. See par.
xlim, ylim	Limits of the x- and y-axes. See par.

llty.arg	Line type. See the lty argument of par.
col.arg	Line color. See the col argument of par.
llwd.arg	Line width. See the lwd argument of par.
...	Arguments passed into the plot function when setting up the entire plot. Useful arguments here include main and las.

Details

The above graphical parameters offer some flexibility when plotting the quantiles.

Value

The list answer, which has components

newdata	The argument newdata above from the argument list of deplot.lmscreg, or a one-row data frame constructed out of the x0 argument.
y	The argument y.arg above.
density	Vector of the density function values evaluated at y.arg.

Note

While the graphical arguments of this function are useful to the user, this function should not be called directly.

Author(s)

Thomas W. Yee

References

Yee, T. W. (2004). Quantile regression via vector generalized additive models. *Statistics in Medicine*, **23**, 2295–2315.

See Also

deplot.lmscreg.

Examples

```
fit <- vgam(BMI ~ s(age, df = c(4,2)), lms.bcn(zero = 1), bmi.nz)
## Not run:  y = seq(15, 43, by = 0.25)
deplot(fit, x0 = 20, y = y, xlab = "BMI", col = "green", llwd = 2,
       main = "BMI distribution at ages 20 (green), 40 (blue), 60 (orange)")
deplot(fit, x0 = 40, y = y, add = TRUE, col = "blue", llwd = 2)
deplot(fit, x0 = 60, y = y, add = TRUE, col = "orange", llwd = 2) -> aa

names(aa@post$deplot)
aa@post$deplot$newdata
head(aa@post$deplot$y)
head(aa@post$deplot$density)
## End(Not run)
```

`plotdgaitd.vglm` *Plotting the GAITD Combo Density from a GAITD Regression Object*

Description

Given a GAITD regression object, plots the probability mass function.

Usage

```
plotdgaitd(object, ...)
plotdgaitd.vglm(object, ...)
```

Arguments

`object`	A fitted GAITD combo regression, e.g., `gaitdpoisson`.
`...`	Graphical arguments passed into `dgaitdplot`.

Details

This is meant to be a more convenient function for plotting the PMF of the GAITD combo model from a fitted regression model. The fit should be intercept-only and the distribution should have 1 or 2 parameters. Currently it should work for a `gaitdpoisson` fit. As much information as needed such as the special values is extracted from the object and fed into `dgaitdplot`.

Value

Same as `dgaitdplot`.

Note

This function is subject to change.

Author(s)

T. W. Yee.

See Also

`dgaitdplot`, `spikeplot`, `gaitdpoisson`.

Examples

```
## Not run:
example(gaitdpoisson)
gaitpfit2 <-
  vglm(y1 ~ 1, crit = "coef", trace = TRUE, data = gdata,
       gaitdpoisson(a.mix = a.mix, i.mix = i.mix,
                    i.mlm = i.mlm, eq.ap = TRUE, eq.ip = TRUE,
```

```
                    truncate = tvec, max.support = max.support))
plotdgaitd(gaitpfit2)

## End(Not run)
```

---

plotqrrvglm *Model Diagnostic Plots for QRR-VGLMs*

Description

The residuals of a QRR-VGLM are plotted for model diagnostic purposes.

Usage

```
plotqrrvglm(object, rtype = c("response", "pearson", "deviance", "working"),
            ask = FALSE,
            main = paste(Rtype, "residuals vs latent variable(s)"),
            xlab = "Latent Variable",
            I.tolerances = object@control$eq.tolerances, ...)
```

Arguments

`object`	An object of class "qrrvglm".
`rtype`	Character string giving residual type. By default, the first one is chosen.
`ask`	Logical. If `TRUE`, the user is asked to hit the return key for the next plot.
`main`	Character string giving the title of the plot.
`xlab`	Character string giving the x-axis caption.
`I.tolerances`	Logical. This argument is fed into `Coef(object, I.tolerances = I.tolerances)`.
`...`	Other plotting arguments (see `par`).

Details

Plotting the residuals can be potentially very useful for checking that the model fit is adequate.

Value

The original object.

Note

An ordination plot of a QRR-VGLM can be obtained by `lvplot.qrrvglm`.

Author(s)

Thomas W. Yee

References

Yee, T. W. (2004). A new technique for maximum-likelihood canonical Gaussian ordination. *Ecological Monographs*, **74**, 685–701.

See Also

lvplot.qrrvglm, cqo.

Examples

```
## Not run:
# QRR-VGLM on the hunting spiders data
# This is computationally expensive
set.seed(111)  # This leads to the global solution
hspider[, 1:6] <- scale(hspider[, 1:6])  # Standardize environ vars
p1 <- cqo(cbind(Alopacce, Alopcune, Alopfabr, Arctlute, Arctperi,
                Auloalbi, Pardlugu, Pardmont, Pardnigr, Pardpull,
                Trocterr, Zoraspin) ~
          WaterCon + BareSand + FallTwig + CoveMoss + CoveHerb + ReflLux,
          poissonff, data = hspider, Crow1positive = FALSE)
par(mfrow = c(3, 4))
plot(p1, rtype = "response", col = "blue", pch = 4, las = 1, main = "")

## End(Not run)
```

---

plotqtplot.lmscreg   *Quantile Plot for LMS Quantile Regression*

Description

Plots the quantiles associated with a LMS quantile regression.

Usage

```
plotqtplot.lmscreg(fitted.values, object, newdata = NULL,
    percentiles = object@misc$percentiles, lp = NULL,
    add.arg = FALSE, y = if (length(newdata)) FALSE else TRUE,
    spline.fit = FALSE, label = TRUE, size.label = 0.06,
    xlab = NULL, ylab = "",
    pch = par()$pch, pcex = par()$cex, pcol.arg = par()$col,
    xlim = NULL, ylim = NULL,
    llty.arg = par()$lty, lcol.arg = par()$col, llwd.arg = par()$lwd,
    tcol.arg = par()$col, tadj = 1, ...)
```

Arguments

`fitted.values`	Matrix of fitted values.
`object`	A **VGAM** quantile regression model, i.e., an object produced by modelling functions such as `vglm` and `vgam` with a family function beginning with `"lms."`, e.g., `lms.yjn`.
`newdata`	Data frame at which predictions are made. By default, the original data are used.
`percentiles`	Numerical vector with values between 0 and 100 that specify the percentiles (quantiles). The default is to use the percentiles when fitting the model. For example, the value 50 corresponds to the median.
`lp`	Length of `percentiles`.
`add.arg`	Logical. Add the quantiles to an existing plot?
`y`	Logical. Add the response as points to the plot?
`spline.fit`	Logical. Add a spline curve to the plot?
`label`	Logical. Add the percentiles (as text) to the plot?
`size.label`	Numeric. How much room to leave at the RHS for the label. It is in percent (of the range of the primary variable).
`xlab`	Caption for the x-axis. See `par`.
`ylab`	Caption for the x-axis. See `par`.
`pch`	Plotting character. See `par`.
`pcex`	Character expansion of the points. See `par`.
`pcol.arg`	Color of the points. See the `col` argument of `par`.
`xlim`	Limits of the x-axis. See `par`.
`ylim`	Limits of the y-axis. See `par`.
`llty.arg`	Line type. Line type. See the `lty` argument of `par`.
`lcol.arg`	Color of the lines. See the `col` argument of `par`.
`llwd.arg`	Line width. See the `lwd` argument of `par`.
`tcol.arg`	Color of the text (if `label` is `TRUE`). See the `col` argument of `par`.
`tadj`	Text justification. See the `adj` argument of `par`.
`...`	Arguments passed into the `plot` function when setting up the entire plot. Useful arguments here include `main` and `las`.

Details

The above graphical parameters offer some flexibility when plotting the quantiles.

Value

The matrix of fitted values.

Note

While the graphical arguments of this function are useful to the user, this function should not be called directly.

Author(s)

Thomas W. Yee

References

Yee, T. W. (2004). Quantile regression via vector generalized additive models. *Statistics in Medicine*, **23**, 2295–2315.

See Also

qtplot.lmscreg.

Examples

```
## Not run:
fit <- vgam(BMI ~ s(age, df = c(4,2)), lms.bcn(zero = 1), data = bmi.nz)
qtplot(fit)
qtplot(fit, perc = c(25,50,75,95), lcol = "blue", tcol = "blue", llwd = 2)

## End(Not run)
```

---

plotrcim0 *Main Effects Plot for a Row-Column Interaction Model (RCIM)*

Description

Produces a main effects plot for Row-Column Interaction Models (RCIMs).

Usage

```
plotrcim0(object, centered = TRUE, which.plots = c(1, 2),
          hline0 = TRUE, hlty = "dashed", hcol = par()$col, hlwd = par()$lwd,
          rfirst = 1, cfirst = 1,
          rtype = "h", ctype = "h",
          rcex.lab = 1, rcex.axis = 1, rtick = FALSE,
          ccex.lab = 1, ccex.axis = 1, ctick = FALSE,
          rmain = "Row effects", rsub = "",
          rxlab = "", rylab = "Row effects",
          cmain = "Column effects", csub = "",
          cxlab= "", cylab = "Column effects",
          rcol = par()$col, ccol = par()$col,
          no.warning = FALSE, ...)
```

Arguments

- `object` — An `rcim` object. This should be of rank-0, i.e., main effects only and no interactions.
- `which.plots` — Numeric, describing which plots are to be plotted. The row effects plot is 1 and the column effects plot is 2. Set the value `0`, say, for no plots at all.
- `centered` — Logical. If `TRUE` then the row and column effects are centered (but not scaled) by `scale`. If `FALSE` then the raw effects are used (of which the first are zero by definition).
- `hline0, hlty, hcol, hlwd` — `hline0` is logical. If `TRUE` then a horizontal line is plotted at 0 and the other arguments describe this line. Probably having `hline0 = TRUE` only makes sense when `centered = TRUE`.
- `rfirst, cfirst` — `rfirst` is the level of row that is placed first in the row effects plot, etc.
- `rmain, cmain` — Character. `rmain` is the main label in the row effects plot, etc.
- `rtype, ctype, rsub, csub` — See the `type` and `sub` arguments of `plot.default`.
- `rxlab, rylab, cxlab, cylab` — Character. For the row effects plot, `rxlab` is `xlab` and `rylab` is `ylab`; see `par`. Ditto for `cxlab` and `cylab` for the column effects plot.
- `rcex.lab, ccex.lab` — Numeric. `rcex.lab` is `cex` for the row effects plot label, etc.
- `rcex.axis, ccex.axis` — Numeric. `rcex.axis` is the `cex` argument for the row effects axis label, etc.
- `rtick, ctick` — Logical. If `rtick = TRUE` then add ticks to the row effects plot, etc.
- `rcol, ccol` — `rcol` give a colour for the row effects plot, etc.
- `no.warning` — Logical. If `TRUE` then no warning is issued if the model is not rank-0.
- `...` — Arguments fed into `plot.default`, etc.

Details

This function plots the row and column effects of a rank-0 RCIM. As the result is a main effects plot of a regression analysis, its interpretation when `centered = FALSE` is relative to the baseline (reference level) of a row and column, and should also be considered in light of the link function used. Many arguments that start with `"r"` refer to the row effects plot, and `"c"` for the column effects plot.

Value

The original object with the `post` slot assigned additional information from the plot.

Note

This function should be only used to plot the object of rank-0 RCIM. If the rank is positive then it will issue a warning.

Using an argument `ylim` will mean the row and column effects are plotted on a common scale; see `plot.window`.

Author(s)

T. W. Yee, A. F. Hadi.

See Also

moffset Rcim, rcim.

Examples

```
alcoff.e <- moffset(alcoff, "6", "Mon", postfix = "*")  # Effective day
fit0 <- rcim(alcoff.e, family = poissonff)
## Not run: par(oma = c(0, 0, 4, 0), mfrow = 1:2)  # For all plots below too
ii <- plot(fit0, rcol = "blue", ccol = "orange",
           lwd = 4, ylim = c(-2, 2),  # A common ylim
           cylab = "Effective daily effects", rylab = "Hourly effects",
           rxlab = "Hour", cxlab = "Effective day")
ii@post  # Endowed with additional information

## End(Not run)

# Negative binomial example
## Not run:
fit1 <- rcim(alcoff.e, negbinomial, trace = TRUE)
plot(fit1, ylim = c(-2, 2))
## End(Not run)

# Univariate normal example
fit2 <- rcim(alcoff.e, uninormal, trace = TRUE)
## Not run:  plot(fit2, ylim = c(-200, 400))

# Median-polish example
## Not run:
fit3 <- rcim(alcoff.e, alaplace1(tau = 0.5), maxit = 1000, trace = FALSE)
plot(fit3, ylim = c(-200, 250))
## End(Not run)

# Zero-inflated Poisson example on "crashp" (no 0s in alcoff)
## Not run:
cbind(rowSums(crashp))  # Easy to see the data
cbind(colSums(crashp))  # Easy to see the data
fit4 <- rcim(Rcim(crashp, rbaseline = "5", cbaseline = "Sun"),
             zipoissonff, trace = TRUE)
plot(fit4, ylim = c(-3, 3))
## End(Not run)
```

---

plotvgam *Default VGAM Plotting*

Description

Component functions of a `vgam-class` object can be plotted with `plotvgam()`. These are on the scale of the linear/additive predictor.

Usage

```
plotvgam(x, newdata = NULL, y = NULL, residuals = NULL,
         rugplot = TRUE, se = FALSE, scale = 0, raw = TRUE,
         offset.arg = 0, deriv.arg = 0, overlay = FALSE,
         type.residuals = c("deviance", "working", "pearson", "response"),
         plot.arg = TRUE, which.term = NULL, which.cf = NULL,
         control = plotvgam.control(...), varxij = 1, ...)
```

Arguments

`x`	A fitted **VGAM** object, e.g., produced by `vgam`, `vglm`, or `rrvglm`.
`newdata`	Data frame. May be used to reconstruct the original data set.
`y`	Unused.
`residuals`	Logical. If `TRUE` then residuals are plotted. See `type.residuals`
`rugplot`	Logical. If `TRUE` then a rug plot is plotted at the foot of each plot. These values are jittered to expose ties.
`se`	Logical. If `TRUE` then approximate ± 2 pointwise standard error bands are included in the plot.
`scale`	Numerical. By default, each plot will have its own y-axis scale. However, by specifying a value, each plot's y-axis scale will be at least `scale` wide.
`raw`	Logical. If `TRUE` then the smooth functions are those obtained directly by the algorithm, and are plotted without having to premultiply with the constraint matrices. If `FALSE` then the smooth functions have been premultiply by the constraint matrices. The `raw` argument is directly fed into `predict.vgam()`.
`offset.arg`	Numerical vector of length r. These are added to the component functions. Useful for separating out the functions when `overlay` is `TRUE`. If `overlay` is `TRUE` and there is one covariate then using the intercept values as the offsets can be a good idea.
`deriv.arg`	Numerical. The order of the derivative. Should be assigned an small integer such as 0, 1, 2. Only applying to `s()` terms, it plots the derivative.
`overlay`	Logical. If `TRUE` then component functions of the same covariate are overlaid on each other. The functions are centered, so `offset.arg` can be useful when `overlay` is `TRUE`.
`type.residuals`	if `residuals` is `TRUE` then the first possible value of this vector, is used to specify the type of residual.
`plot.arg`	Logical. If `FALSE` then no plot is produced.
`which.term`	Character or integer vector containing all terms to be plotted, e.g., `which.term = c("s(age)", "s(height"))` or `which.term = c(2, 5, 9)`. By default, all are plotted.

`which.cf`	An integer-valued vector specifying which linear/additive predictors are to be plotted. The values must be from the set $\{1,2,\ldots,r\}$. By default, all are plotted.
`control`	Other control parameters. See `plotvgam.control`.
`...`	Other arguments that can be fed into `plotvgam.control`. This includes line colors, line widths, line types, etc.
`varxij`	Positive integer. Used if `xij` of `vglm.control` was used, this chooses which inner argument the component is plotted against. This argument is related to `raw = TRUE` and terms such as `NS(dum1, dum2)` and constraint matrices that have more than one column. The default would plot the smooth against `dum1` but setting `varxij = 2` could mean plotting the smooth against `dum2`. See the **VGAM** website for further information.

Details

In this help file M is the number of linear/additive predictors, and r is the number of columns of the constraint matrix of interest.

Many of `plotvgam()`'s options can be found in `plotvgam.control`, e.g., line types, line widths, colors.

Value

The original object, but with the `preplot` slot of the object assigned information regarding the plot.

Note

While `plot(fit)` will work if `class(fit)` is `"vgam"`, it is necessary to use `plotvgam(fit)` explicitly otherwise.

`plotvgam()` is quite buggy at the moment.

Author(s)

Thomas W. Yee

See Also

`vgam`, `plotvgam.control`, `predict.vgam`, `plotvglm`, `vglm`.

Examples

```
coalminers <- transform(coalminers, Age = (age - 42) / 5)
fit <- vgam(cbind(nBnW, nBW, BnW, BW) ~ s(Age),
            binom2.or(zero = NULL), data = coalminers)
## Not run:  par(mfrow = c(1,3))
plot(fit, se = TRUE, ylim = c(-3, 2), las = 1)
plot(fit, se = TRUE, which.cf = 1:2, lcol = "blue", scol = "orange",
     ylim = c(-3, 2))
plot(fit, se = TRUE, which.cf = 1:2, lcol = "blue", scol = "orange",
     overlay = TRUE)
## End(Not run)
```

plotvgam.control *Control Function for plotvgam()*

Description

Provides default values for many arguments available for `plotvgam()`.

Usage

```
plotvgam.control(which.cf = NULL,
    xlim = NULL, ylim = NULL, llty = par()$lty, slty = "dashed",
    pcex = par()$cex, pch = par()$pch, pcol = par()$col,
    lcol = par()$col, rcol = par()$col, scol = par()$col,
    llwd = par()$lwd, slwd = par()$lwd, add.arg = FALSE,
    one.at.a.time = FALSE, .include.dots = TRUE, noxmean = FALSE,
    shade = FALSE, shcol = "gray80", main = "", ...)
```

Arguments

`which.cf`	Integer vector specifying which component functions are to be plotted (for each covariate). Must have values from the set $\{1,2,\ldots,M\}$.
`xlim`	Range for the x-axis.
`ylim`	Range for the y-axis.
`llty`	Line type for the fitted functions (lines). Fed into `par(lty)`.
`slty`	Line type for the standard error bands. Fed into `par(lty)`.
`pcex`	Character expansion for the points (residuals). Fed into `par(cex)`.
`pch`	Character used for the points (residuals). Same as `par(pch)`.
`pcol`	Color of the points. Fed into `par(col)`.
`lcol`	Color of the fitted functions (lines). Fed into `par(col)`.
`rcol`	Color of the rug plot. Fed into `par(col)`.
`scol`	Color of the standard error bands. Fed into `par(col)`.
`llwd`	Line width of the fitted functions (lines). Fed into `par(lwd)`.
`slwd`	Line width of the standard error bands. Fed into `par(lwd)`.
`add.arg`	Logical. If `TRUE` then the plot will be added to an existing plot, otherwise a new plot will be made.
`one.at.a.time`	Logical. If `TRUE` then the plots are done one at a time, with the user having to hit the return key between the plots.
`.include.dots`	Not to be used by the user.
`noxmean`	Logical. If `TRUE` then the point at the mean of x, which is added when standard errors are specified and it thinks the function is linear, is not added. One might use this argument if `ylab` is specified.

shade, shcol	shade is logical; if TRUE then the pointwise SE band is shaded gray by default. The colour can be adjusted by setting shcol. These arguments are ignored unless se = TRUE and overlay = FALSE; If shade = TRUE then scol is ignored.
main	Character vector, recycled to the number needed.
...	Other arguments that may be fed into par().

In the above, M is the number of linear/additive predictors.

Details

The most obvious features of plotvgam can be controlled by the above arguments.

Value

A list with values matching the arguments.

Author(s)

Thomas W. Yee

References

Yee, T. W. and Wild, C. J. (1996). Vector generalized additive models. *Journal of the Royal Statistical Society, Series B, Methodological*, **58**, 481–493.

See Also

plotvgam.

Examples

```
plotvgam.control(lcol = c("red", "blue"), scol = "darkgreen", se = TRUE)
```

---

plotvglm *Plots for VGLMs*

Description

Currently this function plots the Pearson residuals versus the linear predictors (M plots) and plots the Pearson residuals versus the hat values (M plots).

Usage

```
plotvglm(x, which = "(All)", ...)
```

Arguments

x	An object of class "vglm" (see vglm-class) or inherits from that class.
which	If a subset of the plots is required, specify a subset of the numbers 1:(2*M). The default is to plot them all.
...	Arguments fed into the primitive plot functions.

Details

This function is under development. Currently it plots the Pearson residuals against the predicted values (on the transformed scale) and the hat values. There are $2M$ plots in total, therefore users should call par to assign, e.g., the mfrow argument. Note: Section 3.7 of Yee (2015) describes the Pearson residuals and hat values for VGLMs.

Value

Returns the object invisibly.

Author(s)

T. W. Yee

See Also

plotvgam, plotvgam.control, vglm.

Examples

```
## Not run:
ndata <- data.frame(x2 = runif(nn <- 200))
ndata <- transform(ndata, y1 = rnbinom(nn, mu = exp(3+x2), size = exp(1)))
fit1 <- vglm(y1 ~ x2, negbinomial, data = ndata, trace = TRUE)
coef(fit1, matrix = TRUE)
par(mfrow = c(2, 2))
plot(fit1)

# Manually produce the four plots
plot(fit1, which = 1, col = "blue", las = 1, main = "main1")
abline(h = 0, lty = "dashed", col = "gray50")
plot(fit1, which = 2, col = "blue", las = 1, main = "main2")
abline(h = 0, lty = "dashed", col = "gray50")
plot(fit1, which = 3, col = "blue", las = 1, main = "main3")
plot(fit1, which = 4, col = "blue", las = 1, main = "main4")

## End(Not run)
```

`pneumo` *Pneumoconiosis in Coalminers Data*

Description

The `pneumo` data frame has 8 rows and 4 columns. Exposure time is explanatory, and there are 3 ordinal response variables.

Usage

```
data(pneumo)
```

Format

This data frame contains the following columns:

exposure.time a numeric vector, in years

normal a numeric vector, counts

mild a numeric vector, counts

severe a numeric vector, counts

Details

These were collected from coalface workers. In the original data set, the two most severe categories were combined.

Source

Ashford, J.R., 1959. An approach to the analysis of data for semi-quantal responses in biological assay. *Biometrics*, **15**, 573–581.

References

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*, 2nd ed. London: Chapman & Hall.

See Also

`cumulative`.

Examples

```
# Fit the proportional odds model, p.179, in McCullagh and Nelder (1989)
pneumo <- transform(pneumo, let = log(exposure.time))
vglm(cbind(normal, mild, severe) ~ let, propodds, data = pneumo)
```

poisson.points *Poisson-points-on-a-plane/volume Distances Distribution*

Description

Estimating the density parameter of the distances from a fixed point to the u-th nearest point, in a plane or volume.

Usage

```
poisson.points(ostatistic, dimension = 2, link = "loglink",
               idensity = NULL, imethod = 1)
```

Arguments

`ostatistic`	Order statistic. A single positive value, usually an integer. For example, the value 5 means the response are the distances of the fifth nearest value to that point (usually over many planes or volumes). Non-integers are allowed because the value 1.5 coincides with `maxwell` when `dimension = 2`. Note: if `ostatistic = 1` and `dimension = 2` then this **VGAM** family function coincides with `rayleigh`.
`dimension`	The value 2 or 3; 2 meaning a plane and 3 meaning a volume.
`link`	Parameter link function applied to the (positive) density parameter, called λ below. See `Links` for more choices.
`idensity`	Optional initial value for the parameter. A `NULL` value means a value is obtained internally. Use this argument if convergence failure occurs.
`imethod`	An integer with value `1` or `2` which specifies the initialization method for λ. If failure to converge occurs try another value and/or else specify a value for `idensity`.

Details

Suppose the number of points in any region of area A of the plane is a Poisson random variable with mean λA (i.e., λ is the *density* of the points). Given a fixed point P, define D_1, D_2,... to be the distance to the nearest point to P, second nearest to P, etc. This **VGAM** family function estimates λ since the probability density function for D_u is easily derived, $u = 1, 2, \ldots$. Here, u corresponds to the argument `ostatistic`.

Similarly, suppose the number of points in any volume V is a Poisson random variable with mean λV where, once again, λ is the *density* of the points. This **VGAM** family function estimates λ by specifying the argument `ostatistic` and using `dimension = 3`.

The mean of D_u is returned as the fitted values. Newton-Raphson is the same as Fisher-scoring.

Value

An object of class "vglmff" (see `vglmff-class`). The object is used by modelling functions such as `vglm`, `rrvglm` and `vgam`.

Warning

Convergence may be slow if the initial values are far from the solution. This often corresponds to the situation when the response values are all close to zero, i.e., there is a high density of points.

Formulae such as the means have not been fully checked.

Author(s)

T. W. Yee

See Also

poissonff, maxwell, rayleigh.

Examples

```
pdata <- data.frame(y = rgamma(10, shape = exp(-1)))  # Not proper data!
ostat <- 2
fit <- vglm(y ~ 1, poisson.points(ostat, 2), data = pdata,
            trace = TRUE, crit = "coef")
fit <- vglm(y ~ 1, poisson.points(ostat, 3), data = pdata,
            trace = TRUE, crit = "coef")  # Slow convergence?
fit <- vglm(y ~ 1, poisson.points(ostat, 3, idensi = 1), data = pdata,
            trace = TRUE, crit = "coef")
head(fitted(fit))
with(pdata, mean(y))
coef(fit, matrix = TRUE)
Coef(fit)
```

---

poissonff *Poisson Regression*

Description

Family function for a generalized linear model fitted to Poisson responses.

Usage

```
poissonff(link = "loglink", imu = NULL, imethod = 1,
          parallel = FALSE, zero = NULL, bred = FALSE,
          earg.link = FALSE, type.fitted = c("mean", "quantiles"),
          percentiles = c(25, 50, 75))
```

Arguments

- `link` Link function applied to the mean or means. See `Links` for more choices and information.
- `parallel` A logical or formula. Used only if the response is a matrix.
- `imu, imethod` See `CommonVGAMffArguments` for more information.
- `zero` Can be an integer-valued vector specifying which linear/additive predictors are modelled as intercepts only. The values must be from the set $\{1,2,\ldots,M\}$, where M is the number of columns of the matrix response. See `CommonVGAMffArguments` for more information.
- `bred, earg.link` Details at `CommonVGAMffArguments`. Setting `bred = TRUE` should work for multiple responses and all **VGAM** link functions; it has been tested for `loglink`, `identity` but further testing is required.
- `type.fitted, percentiles` Details at `CommonVGAMffArguments`.

Details

M defined above is the number of linear/additive predictors. With overdispersed data try `negbinomial`.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, `vgam`, `rrvglm`, `cqo`, and `cao`.

Warning

With multiple responses, assigning a known dispersion parameter for *each* response is not handled well yet. Currently, only a single known dispersion parameter is handled well.

Note

This function will handle a matrix response automatically.

Regardless of whether the dispersion parameter is to be estimated or not, its value can be seen from the output from the `summary()` of the object.

Author(s)

Thomas W. Yee

References

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*, 2nd ed. London: Chapman & Hall.

See Also

Links, hdeff.vglm, negbinomial, genpoisson1, genpoisson2, genpoisson0, gaitdpoisson, zipoisson, pospoisson, oipospoisson, otpospoisson, skellam, mix2poisson, cens.poisson, ordpoisson, amlpoisson, inv.binomial, simulate.vlm, loglink, polf, rrvglm, cqo, cao, binomialff, poisson, Poisson, poisson.points, ruge, V1, V2, residualsvglm, margeff.

Examples

```
poissonff()

set.seed(123)
pdata <- data.frame(x2 = rnorm(nn <- 100))
pdata <- transform(pdata, y1 = rpois(nn, exp(1 + x2)),
                          y2 = rpois(nn, exp(1 + x2)))
(fit1 <- vglm(cbind(y1, y2) ~ x2, poissonff, data = pdata))
(fit2 <- vglm(y1 ~ x2, poissonff(bred = TRUE), data = pdata))
coef(fit1, matrix = TRUE)
coef(fit2, matrix = TRUE)

nn <- 200
cdata <- data.frame(x2 = rnorm(nn), x3 = rnorm(nn), x4 = rnorm(nn))
cdata <- transform(cdata, lv1 = 0 + x3 - 2*x4)
cdata <- transform(cdata, lambda1 = exp(3 - 0.5 *  (lv1-0)^2),
                          lambda2 = exp(2 - 0.5 *  (lv1-1)^2),
                          lambda3 = exp(2 - 0.5 * ((lv1+4)/2)^2))
cdata <- transform(cdata, y1 = rpois(nn, lambda1),
                          y2 = rpois(nn, lambda2),
                          y3 = rpois(nn, lambda3))
## Not run:  lvplot(p1, y = TRUE, lcol = 2:4, pch = 2:4, pcol = 2:4, rug = FALSE)
```

PoissonPoints *Poisson Points Distribution*

Description

Density for the PoissonPoints distribution.

Usage

```
dpois.points(x, lambda, ostatistic, dimension = 2, log = FALSE)
```

Arguments

x	vector of quantiles.
lambda	the mean density of points.
ostatistic	positive values, usually integers.
dimension	Either 2 and/or 3.
log	Logical; if TRUE, the logarithm is returned.

Details

See poisson.points, the **VGAM** family function for estimating the parameters, for the formula of the probability density function and other details.

Value

dpois.points gives the density.

See Also

poisson.points, dpois, Maxwell.

Examples

```
## Not run:  lambda <- 1; xvec <- seq(0, 2, length = 400)
plot(xvec, dpois.points(xvec, lambda, ostat = 1, dimension = 2),
     type = "l", las = 1, col = "blue",
     sub = "First order statistic",
     main = paste("PDF of PoissonPoints distribution with lambda = ",
                  lambda, " and on the plane", sep = ""))
## End(Not run)
```

Polono *The Poisson Lognormal Distribution*

Description

Density, distribution function and random generation for the Poisson lognormal distribution.

Usage

```
dpolono(x, meanlog = 0, sdlog = 1, bigx = 170, ...)
ppolono(q, meanlog = 0, sdlog = 1,
        isOne = 1 - sqrt( .Machine$double.eps ), ...)
rpolono(n, meanlog = 0, sdlog = 1)
```

Arguments

x, q	vector of quantiles.
n	number of observations. If length(n) > 1 then the length is taken to be the number required.
meanlog, sdlog	the mean and standard deviation of the normal distribution (on the log scale). They match the arguments in Lognormal.

bigx	Numeric. This argument is for handling large values of x and/or when integrate fails. A first order Taylor series approximation [Equation (7) of Bulmer (1974)] is used at values of x that are greater or equal to this argument. For bigx = 10, he showed that the approximation has a relative error less than 0.001 for values of meanlog and sdlog "likely to be encountered in practice". The argument can be assigned Inf in which case the approximation is not used.
isOne	Used to test whether the cumulative probabilities have effectively reached unity.
...	Arguments passed into integrate.

Details

The Poisson lognormal distribution is similar to the negative binomial in that it can be motivated by a Poisson distribution whose mean parameter comes from a right skewed distribution (gamma for the negative binomial and lognormal for the Poisson lognormal distribution).

Value

dpolono gives the density, ppolono gives the distribution function, and rpolono generates random deviates.

Note

By default, dpolono involves numerical integration that is performed using integrate. Consequently, computations are very slow and numerical problems may occur (if so then the use of ... may be needed). Alternatively, for extreme values of x, meanlog, sdlog, etc., the use of bigx = Inf avoids the call to integrate, however the answer may be a little inaccurate.

For the maximum likelihood estimation of the 2 parameters a **VGAM** family function called polono(), say, has not been written yet.

Author(s)

T. W. Yee. Some anonymous soul kindly wrote ppolono() and improved the original dpolono().

References

Bulmer, M. G. (1974). On fitting the Poisson lognormal distribution to species-abundance data. *Biometrics*, **30**, 101–110.

See Also

lognormal, poissonff, negbinomial.

Examples

```
meanlog <- 0.5; sdlog <- 0.5; yy <- 0:19
sum(proby <- dpolono(yy, m = meanlog, sd = sdlog))  # Should be 1
max(abs(cumsum(proby) - ppolono(yy, m = meanlog, sd = sdlog)))  # Should be 0

## Not run:  opar = par(no.readonly = TRUE)
par(mfrow = c(2, 2))
```

```
plot(yy, proby, type = "h", col = "blue", ylab = "P[Y=y]", log = "",
     main = paste("Poisson lognormal(m = ", meanlog,
                  ", sdl = ", sdlog, ")", sep = ""))

y <- 0:190  # More extreme values; use the approximation and plot on a log scale
(sum(proby <- dpolono(y, m = meanlog, sd = sdlog, bigx = 100)))  # Should be 1
plot(y, proby, type = "h", col = "blue", ylab = "P[Y=y] (log)", log = "y",
     main = paste("Poisson lognormal(m = ", meanlog,
                  ", sdl = ", sdlog, ")", sep = ""))  # Note the kink at bigx

# Random number generation
table(y <- rpolono(n = 1000, m = meanlog, sd = sdlog))
hist(y, breaks = ((-1):max(y))+0.5, prob = TRUE, border = "blue")
par(opar)
## End(Not run)
```

---

posbernoulli.b *Positive Bernoulli Family Function with Behavioural Effects*

Description

Fits a GLM-/GAM-like model to multiple Bernoulli responses where each row in the capture history matrix response has at least one success (capture). Capture history behavioural effects are accommodated.

Usage

```
posbernoulli.b(link = "logitlink", drop.b = FALSE ~ 1,
   type.fitted = c("likelihood.cond", "mean.uncond"), I2 = FALSE,
   ipcapture = NULL, iprecapture = NULL,
   p.small = 1e-4, no.warning = FALSE)
```

Arguments

`link, drop.b, ipcapture, iprecapture`
: See `CommonVGAMffArguments` for information about these arguments. By default the parallelism assumption does not apply to the intercept. With an intercept-only model setting `drop.b = TRUE ~ 1` results in the M_0/M_h model.

`I2`
: Logical. This argument is used for terms that are not parallel. If `TRUE` then the constraint matrix `diag(2)` (the general default constraint matrix in **VGAM**) is used, else `cbind(0:1, 1)`. The latter means the first element/column corresponds to the behavioural effect. Consequently it and its standard error etc. can be accessed directly without subtracting two quantities.

`type.fitted`
: Details at `posbernoulli.tb`.

`p.small, no.warning`
: See `posbernoulli.t`.

Details

This model (commonly known as M_b/M_{bh} in the capture–recapture literature) operates on a capture history matrix response of 0s and 1s ($n \times \tau$). See posbernoulli.t for details, e.g., common assumptions with other models. Once an animal is captured for the first time, it is marked/tagged so that its future capture history can be recorded. The effect of the recapture probability is modelled through a second linear/additive predictor. It is well-known that some species of animals are affected by capture, e.g., trap-shy or trap-happy. This **VGAM** family function *does* allow the capture history to be modelled via such behavioural effects. So does posbernoulli.tb but posbernoulli.t cannot.

The number of linear/additive predictors is $M = 2$, and the default links are $(logit\, p_c, logit\, p_r)^T$ where p_c is the probability of capture and p_r is the probability of recapture. The fitted value returned is of the same dimension as the response matrix, and depends on the capture history: prior to being first captured, it is `pcapture`. Afterwards, it is `precapture`.

By default, the constraint matrices for the intercept term and the other covariates are set up so that p_r differs from p_c by a simple binary effect, on a logit scale. However, this difference (the behavioural effect) is more directly estimated by having `I2 = FALSE`. Then it allows an estimate of the trap-happy/trap-shy effect; these are positive/negative values respectively. If `I2 = FALSE` then the (nonstandard) constraint matrix used is `cbind(0:1, 1)`, meaning the first element can be interpreted as the behavioural effect.

Value

An object of class `"vglmff"` (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Note

The dependent variable is *not* scaled to row proportions. This is the same as posbernoulli.t and posbernoulli.tb but different from posbinomial and binomialff.

Author(s)

Thomas W. Yee.

References

See posbernoulli.t.

See Also

posbernoulli.t and posbernoulli.tb (including estimating N), deermice, dposbern, rposbern, posbinomial, aux.posbernoulli.t, prinia.

Examples

```
# deermice data ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,

# Fit a M_b model
M.b <- vglm(cbind(y1, y2, y3, y4, y5, y6) ~ 1,
```

```
            posbernoulli.b, data = deermice, trace = TRUE)
coef(M.b)["(Intercept):1"]  # Behavioural effect on logit scale
coef(M.b, matrix = TRUE)
constraints(M.b, matrix = TRUE)
summary(M.b, presid = FALSE)

# Fit a M_bh model
M.bh <- vglm(cbind(y1, y2, y3, y4, y5, y6) ~ sex + weight,
             posbernoulli.b, data = deermice, trace = TRUE)
coef(M.bh, matrix = TRUE)
coef(M.bh)["(Intercept):1"]  # Behavioural effect on logit scale
# (2,1) elt is for the behavioural effect:
constraints(M.bh)[["(Intercept)"]]
summary(M.bh, presid = FALSE)  # Significant trap-happy effect
# Approx. 95 percent confidence for the behavioural effect:
SE.M.bh <- coef(summary(M.bh))["(Intercept):1", "Std. Error"]
coef(M.bh)["(Intercept):1"] + c(-1, 1) * 1.96 * SE.M.bh

# Fit a M_h model
M.h <- vglm(cbind(y1, y2, y3, y4, y5, y6) ~ sex + weight,
            posbernoulli.b(drop.b = TRUE ~ sex + weight),
            data = deermice, trace = TRUE)
coef(M.h, matrix = TRUE)
constraints(M.h, matrix = TRUE)
summary(M.h, presid = FALSE)

# Fit a M_0 model
M.0 <- vglm(cbind(    y1 + y2 + y3 + y4 + y5 + y6,
                  6 - y1 - y2 - y3 - y4 - y5 - y6) ~ 1,
            posbinomial, data = deermice, trace = TRUE)
coef(M.0, matrix = TRUE)
summary(M.0, presid = FALSE)


# Simulated data set ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
set.seed(123); nTimePts <- 5; N <- 1000  # N is the popn size
pdata <- rposbern(N, nTimePts=nTimePts, pvars=2, is.popn=TRUE)
nrow(pdata)  # < N (because some animals were never captured)
# The truth: xcoeffs are c(-2, 1, 2) and cap.effect = +1

M.bh.2 <- vglm(cbind(y1, y2, y3, y4, y5) ~ x2,
               posbernoulli.b, data = pdata, trace = TRUE)
coef(M.bh.2)
coef(M.bh.2, matrix = TRUE)
constraints(M.bh.2, matrix = TRUE)
summary(M.bh.2, presid = FALSE)
head(depvar(M.bh.2))    # Capture history response matrix
head(M.bh.2@extra$cap.hist1)  # Info on its capture history
head(M.bh.2@extra$cap1)  # When it was first captured
head(fitted(M.bh.2))     # Depends on capture history
(trap.effect <- coef(M.bh.2)["(Intercept):1"])  # Should be +1
head(model.matrix(M.bh.2, type = "vlm"), 21)
head(pdata)
```

```
summary(pdata)
dim(depvar(M.bh.2))
vcov(M.bh.2)

M.bh.2@extra$N.hat  # Population size estimate; should be about N
M.bh.2@extra$SE.N.hat  # SE of the estimate of the population size
# An approximate 95 percent confidence interval:
round(M.bh.2@extra$N.hat + c(-1, 1)*1.96* M.bh.2@extra$SE.N.hat, 1)
```

`posbernoulli.t` *Positive Bernoulli Family Function with Time Effects*

Description

Fits a GLM/GAM-like model to multiple Bernoulli responses where each row in the capture history matrix response has at least one success (capture). Sampling occasion effects are accommodated.

Usage

```
posbernoulli.t(link = "logitlink", parallel.t = FALSE ~ 1,
    iprob = NULL, p.small = 1e-4, no.warning = FALSE,
    type.fitted = c("probs", "onempall0"))
```

Arguments

`link, iprob, parallel.t`
: See `CommonVGAMffArguments` for information. By default, the parallelism assumption does not apply to the intercept. Setting `parallel.t = FALSE ~ -1`, or equivalently `parallel.t = FALSE ~ 0`, results in the M_0/M_h model.

`p.small, no.warning`
: A small probability value used to give a warning for the Horvitz–Thompson estimator. Any estimated probability value less than `p.small` will result in a warning, however, setting `no.warning = TRUE` will suppress this warning if it occurs. This is because the Horvitz-Thompson estimator is the sum of the reciprocal of such probabilities, therefore any probability that is too close to 0 will result in an unstable estimate.

`type.fitted`
: See `CommonVGAMffArguments` for information. The default is to return a matrix of probabilities. If `"onempall0"` is chosen then the the probability that each animal is captured at least once in the course of the study is returned. The abbreviation stands for one minus the probability of all 0s, and the quantity appears in the denominator of the usual formula.

Details

These models (commonly known as M_t or M_{th} (no prefix h means it is an intercept-only model) in the capture–recapture literature) operate on a capture history matrix response of 0s and 1s ($n \times \tau$). Each column is a sampling occasion where animals are potentially captured (e.g., a field trip), and each row is an individual animal. Capture is a 1, else a 0. No removal of animals from the population is made (closed population), e.g., no immigration or emigration. Each row of the response matrix has at least one capture. Once an animal is captured for the first time, it is marked/tagged so that its future capture history can be recorded. Then it is released immediately back into the population to remix. It is released immediately after each recapture too. It is assumed that the animals are independent and that, for a given animal, each sampling occasion is independent. And animals do not lose their marks/tags, and all marks/tags are correctly recorded.

The number of linear/additive predictors is equal to the number of sampling occasions, i.e., $M = \tau$, say. The default link functions are $(logit\, p_1, \ldots, logit\, p_\tau)^T$ where each p_j denotes the probability of capture at time point j. The fitted value returned is a matrix of probabilities of the same dimension as the response matrix.

A conditional likelihood is maximized here using Fisher scoring. Each sampling occasion has a separate probability that is modelled here. The probabilities can be constrained to be equal by setting `parallel.t = FALSE ~ 0`; then the results are effectively the same as `posbinomial` except the binomial constants are not included in the log-likelihood. If `parallel.t = TRUE ~ 0` then each column should have at least one 1 and at least one 0.

It is well-known that some species of animals are affected by capture, e.g., trap-shy or trap-happy. This **VGAM** family function does *not* allow any behavioral effect to be modelled (`posbernoulli.b` and `posbernoulli.tb` do) because the denominator of the likelihood function must be free of behavioral effects.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Upon fitting the `extra` slot has a (list) component called `N.hat` which is a point estimate of the population size N (it is the Horvitz-Thompson (1952) estimator). And there is a component called `SE.N.hat` containing its standard error.

Note

The `weights` argument of `vglm` need not be assigned, and the default is just a matrix of ones.

Fewer numerical problems are likely to occur for `parallel.t = TRUE`. Data-wise, each sampling occasion may need at least one success (capture) and one failure. Less stringent conditions in the data are needed when `parallel.t = TRUE`. Ditto when parallelism is applied to the intercept too.

The response matrix is returned unchanged; i.e., not converted into proportions like `posbinomial`. If the response matrix has column names then these are used in the labelling, else `prob1`, `prob2`, etc. are used.

Using `AIC()` or `BIC()` to compare `posbernoulli.t`, `posbernoulli.b`, `posbernoulli.tb` models with a `posbinomial` model requires `posbinomial(omit.constant = TRUE)` because one needs to remove the normalizing constant from the log-likelihood function. See `posbinomial` for an example.

Author(s)

Thomas W. Yee.

References

Huggins, R. M. (1991). Some practical aspects of a conditional likelihood approach to capture experiments. *Biometrics*, **47**, 725–732.

Huggins, R. M. and Hwang, W.-H. (2011). A review of the use of conditional likelihood in capture–recapture experiments. *International Statistical Review*, **79**, 385–400.

Otis, D. L. and Burnham, K. P. and White, G. C. and Anderson, D. R. (1978). Statistical inference from capture data on closed animal populations, *Wildlife Monographs*, **62**, 3–135.

Yee, T. W. and Stoklosa, J. and Huggins, R. M. (2015). The **VGAM** package for capture–recapture data using the conditional likelihood. *Journal of Statistical Software*, **65**, 1–33. doi:10.18637/jss.v065.i05.

See Also

posbernoulli.b, posbernoulli.tb, Select, deermice, Huggins89table1, Huggins89.t1, dposbern, rposbern, posbinomial, AICvlm, BICvlm, prinia.

Examples

```
M.t <- vglm(cbind(y1, y2, y3, y4, y5, y6) ~ 1, posbernoulli.t,
            data = deermice, trace = TRUE)
coef(M.t, matrix = TRUE)
constraints(M.t, matrix = TRUE)
summary(M.t, presid = FALSE)

M.h.1 <- vglm(Select(deermice, "y") ~ sex + weight, trace = TRUE,
              posbernoulli.t(parallel.t = FALSE ~ -1), deermice)
coef(M.h.1, matrix = TRUE)
constraints(M.h.1)
summary(M.h.1, presid = FALSE)
head(depvar(M.h.1))  # Response capture history matrix
dim(depvar(M.h.1))

M.th.2 <- vglm(cbind(y1, y2, y3, y4, y5, y6) ~ sex + weight,
               posbernoulli.t(parallel.t = FALSE), deermice)
# Test the parallelism assumption wrt sex and weight:
lrtest(M.h.1, M.th.2)
coef(M.th.2)
coef(M.th.2, matrix = TRUE)
constraints(M.th.2)
summary(M.th.2, presid = FALSE)
head(model.matrix(M.th.2, type = "vlm"), 21)

M.th.2@extra$N.hat  # Population size estimate; should be about N
M.th.2@extra$SE.N.hat  # SE of the estimate of the population size
# An approximate 95 percent confidence interval:
round(M.th.2@extra$N.hat + c(-1, 1)*1.96* M.th.2@extra$SE.N.hat, 1)

# Fit a M_h model, effectively the parallel M_t model:
deermice <- transform(deermice, ysum = y1 + y2 + y3 + y4 + y5 + y6,
                                tau  = 6)
```

```
M.h.3 <- vglm(cbind(ysum, tau - ysum) ~ sex + weight,
              posbinomial(omit.constant = TRUE), data = deermice)
max(abs(coef(M.h.1) - coef(M.h.3)))  # Should be zero
# Difference is due to the binomial constants:
logLik(M.h.3) - logLik(M.h.1)
```

---

posbernoulli.tb   *Positive Bernoulli Family Function with Time and Behavioural Effects*

Description

Fits a GLM/GAM-like model to multiple Bernoulli responses where each row in the capture history matrix response has at least one success (capture). Sampling occasion effects and behavioural effects are accommodated.

Usage

```
posbernoulli.tb(link = "logitlink", parallel.t = FALSE ~ 1,
   parallel.b = FALSE ~ 0, drop.b = FALSE ~ 1,
   type.fitted = c("likelihood.cond", "mean.uncond"),
   imethod = 1, iprob = NULL,
   p.small = 1e-4, no.warning = FALSE,
   ridge.constant = 0.0001, ridge.power = -4)
```

Arguments

`link, imethod, iprob`
: See `CommonVGAMffArguments` for information.

`parallel.t, parallel.b, drop.b`
: A logical, or formula with a logical as the response. See `CommonVGAMffArguments` for information. The `parallel.`-type arguments specify whether the constraint matrices have a parallelism assumption for the temporal and behavioural effects. Argument `parallel.t` means parallel with respect to time, and matches the same argument name in `posbernoulli.t`.

  Suppose the model is intercept-only. Setting `parallel.t = FALSE ~ 0` results in the M_b model. Setting `drop.b = FALSE ~ 0` results in the M_t model because it drops columns off the constraint matrices corresponding to any behavioural effect. Setting `parallel.t = FALSE ~ 0` and setting `parallel.b = FALSE ~ 0` results in the M_b model. Setting `parallel.t = FALSE ~ 0`, `parallel.b = FALSE ~ 0` and `drop.b = FALSE ~ 0` results in the M_0 model. Note the default for `parallel.t` and `parallel.b` may be unsuitable for data sets which have a large τ because of the large number of parameters; it might be too flexible. If it is desired to have the behaviour affect some of the other covariates then set `drop.b = TRUE ~ 0`.

  The default model has a different intercept for each sampling occasion, a time-parallelism assumption for all other covariates, and a dummy variable representing a single behavioural effect (also in the intercept).

The most flexible model is to set parallel.b = TRUE ~ 0, parallel.t = TRUE ~ 0 and drop.b = TRUE ~ 0. This means that all possible temporal and behavioural effects are estimated, for the intercepts and other covariates. Such a model is *not* recommended; it will contain a lot of paramters.

type.fitted
: Character, one of the choices for the type of fitted value returned. The default is the first one. Partial matching is okay. For "likelihood.cond": the probability defined by the conditional likelihood. For "mean.uncond": the unconditional mean, which should agree with colMeans applied to the response matrix for intercept-only models.

ridge.constant, ridge.power
: Determines the ridge parameters at each IRLS iteration. They are the constant and power (exponent) for the ridge adjustment for the working weight matrices (the capture probability block matrix, hence the first τ diagonal values). At iteration a of the IRLS algorithm a positive value is added to the first τ diagonal elements of the working weight matrices to make them positive-definite. This adjustment is the mean of the diagonal elements of wz multipled by $K \times a^p$ where K is ridge.constant and p is ridge.power. This is always positive but decays to zero as iterations proceed (provided p is negative etc.).

p.small, no.warning
: See posbernoulli.t.

Details

This model (commonly known as M_{tb}/M_{tbh} in the capture–recapture literature) operates on a response matrix of 0s and 1s ($n \times \tau$). See posbernoulli.t for information that is in common. It allows time and behavioural effects to be modelled.

Evidently, the expected information matrix (EIM) seems *not* of full rank (especially in early iterations), so ridge.constant and ridge.power are used to *try* fix up the problem. The default link functions are $(logit\, p_{c1}, \ldots, logit\, p_{c\tau}, logit\, p_{r2}, \ldots, logit\, p_{r\tau})^T$ where the subscript c denotes capture, the subscript r denotes recapture, and it is not possible to recapture the animal at sampling occasion 1. Thus $M = 2\tau - 1$. The parameters are currently prefixed by pcapture and precapture for the capture and recapture probabilities. This **VGAM** family function may be further modified in the future.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Note

It is a good idea to apply the parallelism assumption to each sampling occasion except possibly with respect to the intercepts. Also, a simple behavioural effect such as being modelled using the intercept is recommended; if the behavioural effect is not parallel and/or allowed to apply to other covariates then there will probably be too many parameters, and hence, numerical problems. See M_tbh.1 below.

It is a good idea to monitor convergence. Simpler models such as the M_0/M_h models are best fitted with posbernoulli.t or posbernoulli.b or posbinomial.

Author(s)

Thomas W. Yee.

References

See posbernoulli.t.

See Also

posbernoulli.b (including N.hat), posbernoulli.t, posbinomial, Select, fill1, Huggins89table1, Huggins89.t1, deermice, prinia.

Examples

```
## Not run:
# Example 1: simulated data
nTimePts <- 5  # (aka tau == # of sampling occasions)
nnn <- 1000   # Number of animals
pdata <- rposbern(n = nnn, nTimePts = nTimePts, pvars = 2)
dim(pdata); head(pdata)

M_tbh.1 <- vglm(cbind(y1, y2, y3, y4, y5) ~ x2,
                posbernoulli.tb, data = pdata, trace = TRUE)
coef(M_tbh.1)  # First element is the behavioural effect
coef(M_tbh.1, matrix = TRUE)
constraints(M_tbh.1, matrix = TRUE)
summary(M_tbh.1, presid = FALSE)  # Std errors are approximate
head(fitted(M_tbh.1))
head(model.matrix(M_tbh.1, type = "vlm"), 21)
dim(depvar(M_tbh.1))

M_tbh.2 <- vglm(cbind(y1, y2, y3, y4, y5) ~ x2,
                posbernoulli.tb(parallel.t = FALSE ~  0),
                data = pdata, trace = TRUE)
coef(M_tbh.2)  # First element is the behavioural effect
coef(M_tbh.2, matrix = TRUE)
constraints(M_tbh.2, matrix = TRUE)
summary(M_tbh.2, presid = FALSE)  # Std errors are approximate
head(fitted(M_tbh.2))
head(model.matrix(M_tbh.2, type = "vlm"), 21)
dim(depvar(M_tbh.2))

# Example 2: deermice subset data
fit1 <- vglm(cbind(y1, y2, y3, y4, y5, y6) ~ sex + weight,
             posbernoulli.t, data = deermice, trace = TRUE)
coef(fit1)
coef(fit1, matrix = TRUE)
constraints(fit1, matrix = TRUE)
summary(fit1, presid = FALSE)  # Standard errors are approximate

# fit1 is the same as Fit1 (a M_{th} model):
Fit1 <- vglm(cbind(y1, y2, y3, y4, y5, y6) ~ sex + weight,
```

```
              posbernoulli.tb(drop.b = TRUE ~ sex + weight,
                 parallel.t = TRUE),  # But not for the intercept
              data = deermice, trace = TRUE)
constraints(Fit1)

## End(Not run)
```

posbernUC *Positive Bernoulli Sequence Model*

Description

Density, and random generation for multiple Bernoulli responses where each row in the response matrix has at least one success.

Usage

```
rposbern(n, nTimePts = 5, pvars = length(xcoeff),
  xcoeff = c(-2, 1, 2), Xmatrix = NULL, cap.effect = 1,
  is.popn = FALSE, link = "logitlink", earg.link = FALSE)
dposbern(x, prob, prob0 = prob, log = FALSE)
```

Arguments

x	response vector or matrix. Should only have 0 and 1 values, at least two columns, and each row should have at least one 1.
nTimePts	Number of sampling occasions. Called τ in posbernoulli.b and posbernoulli.t.
n	number of observations. Usually a single positive integer, else the length of the vector is used. See argument is.popn.
is.popn	Logical. If TRUE then argument n is the population size and what is returned may have substantially less rows than n. That is, if an animal has at least one one in its sequence then it is returned, else that animal is not returned because it never was captured.
Xmatrix	Optional **X** matrix. If given, the **X** matrix is not generated internally.
cap.effect	Numeric, the capture effect. Added to the linear predictor if captured previously. A positive or negative value corresponds to a trap-happy and trap-shy effect respectively.
pvars	Number of other numeric covariates that make up the linear predictor. Labelled x1, x2, ..., where the first is an intercept, and the others are independent standard runif random variates. The first pvars elements of xcoeff are used.
xcoeff	The regression coefficients of the linear predictor. These correspond to x1, x2, ..., and the first is for the intercept. The length of xcoeff must be at least pvars.
link, earg.link	The former is used to generate the probabilities for capture at each occasion. Other details at CommonVGAMffArguments.

`prob, prob0` Matrix of probabilities for the numerator and denominators respectively. The default does *not* correspond to the M_b model since the M_b model has a denominator which involves the capture history.

`log` Logical. Return the logarithm of the answer?

Details

The form of the conditional likelihood is described in `posbernoulli.b` and/or `posbernoulli.t` and/or `posbernoulli.tb`. The denominator is equally shared among the elements of the matrix `x`.

Value

`rposbern` returns a data frame with some attributes. The function generates random deviates (τ columns labelled `y1`, `y2`, `...`) for the response. Some indicator columns are also included (those starting with `ch` are for previous capture history). The default setting corresponds to a M_{bh} model that has a single trap-happy effect. Covariates `x1`, `x2`, ... have the same affect on capture/recapture at every sampling occasion (see the argument `parallel.t` in, e.g., `posbernoulli.tb`).

The function `dposbern` gives the density,

Note

The `r`-type function is experimental only and does not follow the usual conventions of `r`-type R functions. It may change a lot in the future. The `d`-type function is more conventional and is less likely to change.

Author(s)

Thomas W. Yee.

See Also

`posbernoulli.tb`, `posbernoulli.b`, `posbernoulli.t`.

Examples

```
rposbern(n = 10)
attributes(pdata <- rposbern(n = 100))
M.bh <- vglm(cbind(y1, y2, y3, y4, y5) ~ x2 + x3,
             posbernoulli.b(I2 = FALSE), pdata, trace = TRUE)
constraints(M.bh)
summary(M.bh)
```

`posbinomial` *Positive Binomial Distribution Family Function*

Description

Fits a positive binomial distribution.

Usage

```
posbinomial(link = "logitlink", multiple.responses = FALSE,
    parallel = FALSE, omit.constant = FALSE, p.small = 1e-4,
    no.warning = FALSE, zero = NULL)
```

Arguments

`link, multiple.responses, parallel, zero`
: Details at `CommonVGAMffArguments`.

`omit.constant`
: Logical. If `TRUE` then the constant (`lchoose(size, size * yprop)` is omitted from the `loglikelihood` calculation. If the model is to be compared using `AIC()` or `BIC()` (see `AICvlm` or `BICvlm`) to the likes of `posbernoulli.tb` etc. then it is important to set `omit.constant = TRUE` because all models then will not have any normalizing constants in the likelihood function. Hence they become comparable. This is because the M_0 Otis et al. (1978) model coincides with `posbinomial()`. See below for an example. Also see `posbernoulli.t` regarding estimating the population size (`N.hat` and `SE.N.hat`) if the number of trials is the same for all observations.

`p.small, no.warning`
: See `posbernoulli.t`.

Details

The positive binomial distribution is the ordinary binomial distribution but with the probability of zero being zero. Thus the other probabilities are scaled up (i.e., divided by $1 - P(Y = 0)$). The fitted values are the ordinary binomial distribution fitted values, i.e., the usual mean.

In the capture–recapture literature this model is called the M_0 if it is an intercept-only model. Otherwise it is called the M_h when there are covariates. It arises from a sum of a sequence of τ-Bernoulli random variates subject to at least one success (capture). Here, each animal has the same probability of capture or recapture, regardless of the τ sampling occasions. Independence between animals and between sampling occasions etc. is assumed.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Warning

Under- or over-flow may occur if the data is ill-conditioned.

Note

The input for this family function is the same as binomialff.

If multiple.responses = TRUE then each column of the matrix response should be a count (the number of successes), and the weights argument should be a matrix of the same dimension as the response containing the number of trials. If multiple.responses = FALSE then the response input should be the same as binomialff.

Yet to be done: a quasi.posbinomial() which estimates a dispersion parameter.

Author(s)

Thomas W. Yee

References

Otis, D. L. et al. (1978). Statistical inference from capture data on closed animal populations, *Wildlife Monographs*, **62**, 3–135.

Patil, G. P. (1962). Maximum likelihood estimation for generalised power series distributions and its application to a truncated binomial distribution. *Biometrika*, **49**, 227–237.

Pearson, K. (1913). *A Monograph on Albinism in Man*. Drapers Company Research Memoirs.

See Also

posbernoulli.b, posbernoulli.t, posbernoulli.tb, binomialff, AICvlm, BICvlm, simulate.vlm.

Examples

```
# Albinotic children in families with 5 kids (from Patil, 1962) ,,,,
albinos <- data.frame(y = c(rep(1, 25), rep(2, 23), rep(3, 10), 4, 5),
                      n = rep(5, 60))
fit1 <- vglm(cbind(y, n-y) ~ 1, posbinomial, albinos, trace = TRUE)
summary(fit1)
Coef(fit1)  # = MLE of p = 0.3088
head(fitted(fit1))
sqrt(vcov(fit1, untransform = TRUE))  # SE = 0.0322

# Fit a M_0 model (Otis et al. 1978) to the deermice data ,,,,,,,,,,
M.0 <- vglm(cbind(    y1 + y2 + y3 + y4 + y5 + y6,
                  6 - y1 - y2 - y3 - y4 - y5 - y6) ~ 1, trace = TRUE,
            posbinomial(omit.constant = TRUE), data = deermice)
coef(M.0, matrix = TRUE)
Coef(M.0)
constraints(M.0, matrix = TRUE)
summary(M.0)
c(   N.hat = M.0@extra$N.hat,     # As tau = 6, i.e., 6 Bernoulli trials
  SE.N.hat = M.0@extra$SE.N.hat)  # per obsn is the same for each obsn

# Compare it to the M_b using AIC and BIC
M.b <- vglm(cbind(y1, y2, y3, y4, y5, y6) ~ 1, trace = TRUE,
            posbernoulli.b, data = deermice)
```

```
sort(c(M.0 = AIC(M.0), M.b = AIC(M.b)))  # Ok since omit.constant=TRUE
sort(c(M.0 = BIC(M.0), M.b = BIC(M.b)))  # Ok since omit.constant=TRUE
```

Posgeom *Positive-Geometric Distribution*

Description

Density, distribution function, quantile function and random generation for the positive-geometric distribution.

Usage

```
dposgeom(x, prob, log = FALSE)
pposgeom(q, prob)
qposgeom(p, prob)
rposgeom(n, prob)
```

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
n	number of observations. Fed into runif.
prob	vector of probabilities of success (of an ordinary geometric distribution). Short vectors are recycled.
log	logical.

Details

The positive-geometric distribution is a geometric distribution but with the probability of a zero being zero. The other probabilities are scaled to add to unity. The mean therefore is $1/prob$.

As $prob$ decreases, the positive-geometric and geometric distributions become more similar. Like similar functions for the geometric distribution, a zero value of prob is not permitted here.

Value

dposgeom gives the density, pposgeom gives the distribution function, qposgeom gives the quantile function, and rposgeom generates random deviates.

Author(s)

T. W. Yee

See Also

zageometric, zigeometric, rgeom.

Examples

```
prob <- 0.75; y <- rposgeom(n = 1000, prob)
table(y)
mean(y)  # Sample mean
1 / prob  # Population mean

(ii <- dposgeom(0:7, prob))
cumsum(ii) - pposgeom(0:7, prob)  # Should be 0s
table(rposgeom(100, prob))

table(qposgeom(runif(1000), prob))
round(dposgeom(1:10, prob) * 1000)  # Should be similar

## Not run:
x <- 0:5
barplot(rbind(dposgeom(x, prob), dgeom(x, prob)),
        beside = TRUE, col = c("blue", "orange"),
        main = paste("Positive geometric(", prob, ") (blue) vs",
        " geometric(", prob, ") (orange)", sep = ""),
        names.arg = as.character(x), las = 1, lwd = 2)
## End(Not run)
```

`posnegbinomial` *Positive Negative Binomial Distribution Family Function*

Description

Maximum likelihood estimation of the two parameters of a positive negative binomial distribution.

Usage

```
posnegbinomial(zero = "size",
   type.fitted = c("mean", "munb", "prob0"),
   mds.min = 0.001, nsimEIM = 500, cutoff.prob = 0.999,
   eps.trig = 1e-07, max.support = 4000, max.chunk.MB = 30,
   lmunb = "loglink", lsize = "loglink", imethod = 1,
   imunb = NULL, iprobs.y = NULL,
   gprobs.y = ppoints(8), isize = NULL,
   gsize.mux = exp(c(-30, -20, -15, -10, -6:3)))
```

Arguments

`lmunb`	Link function applied to the `munb` parameter, which is the mean μ_{nb} of an ordinary negative binomial distribution. See `Links` for more choices.
`lsize`	Parameter link function applied to the dispersion parameter, called `k`. See `Links` for more choices.

isize
: Optional initial value for k, an index parameter. The value 1/k is known as a dispersion parameter. If failure to converge occurs try different values (and/or use imethod). If necessary this vector is recycled to length equal to the number of responses. A value NULL means an initial value for each response is computed internally using a range of values.

nsimEIM, zero, eps.trig
: See CommonVGAMffArguments.

mds.min, iprobs.y, cutoff.prob
: Similar to negbinomial.

imunb, max.support
: Similar to negbinomial.

max.chunk.MB, gsize.mux
: Similar to negbinomial.

imethod, gprobs.y
: See negbinomial.

type.fitted
: See CommonVGAMffArguments for details.

Details

The positive negative binomial distribution is an ordinary negative binomial distribution but with the probability of a zero response being zero. The other probabilities are scaled to sum to unity.

This family function is based on negbinomial and most details can be found there. To avoid confusion, the parameter munb here corresponds to the mean of an ordinary negative binomial distribution negbinomial. The mean of posnegbinomial is

$$\mu_{nb}/(1-p(0))$$

where $p(0) = (k/(k+\mu_{nb}))^k$ is the probability an ordinary negative binomial distribution has a zero value.

The parameters munb and k are not independent in the positive negative binomial distribution, whereas they are in the ordinary negative binomial distribution.

This function handles *multiple* responses, so that a matrix can be used as the response. The number of columns is the number of species, say, and setting zero = -2 means that *all* species have a k equalling a (different) intercept only.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, rrvglm and vgam.

Warning

This family function is fragile; at least two cases will lead to numerical problems. Firstly, the positive-Poisson model corresponds to k equalling infinity. If the data is positive-Poisson or close to positive-Poisson, then the estimated k will diverge to Inf or some very large value. Secondly, if the data is clustered about the value 1 because the munb parameter is close to 0 then numerical problems will also occur. Users should set trace = TRUE to monitor convergence. In the situation when both cases hold, the result returned (which will be untrustworthy) will depend on the initial values.

The negative binomial distribution (NBD) is a strictly unimodal distribution. Any data set that does not exhibit a mode (in the middle) makes the estimation problem difficult. The positive NBD inherits this feature. Set `trace = TRUE` to monitor convergence.

See the example below of a data set where `posbinomial()` fails; the so-called solution is *extremely* poor. This is partly due to a lack of a unimodal shape because the number of counts decreases only. This long tail makes it very difficult to estimate the mean parameter with any certainty. The result too is that the `size` parameter is numerically fraught.

This **VGAM** family function inherits the same warnings as `negbinomial`. And if `k` is much less than 1 then the estimation may be slow.

Note

If the estimated k is very large then fitting a `pospoisson` model is a good idea.

If both `munb` and k are large then it may be necessary to decrease `eps.trig` and increase `max.support` so that the EIMs are positive-definite, e.g., `eps.trig = 1e-8` and `max.support = Inf`.

Author(s)

Thomas W. Yee

References

Barry, S. C. and Welsh, A. H. (2002). Generalized additive modelling and zero inflated count data. *Ecological Modelling*, **157**, 179–188.

Williamson, E. and Bretherton, M. H. (1964). Tables of the logarithmic series distribution. *Annals of Mathematical Statistics*, **35**, 284–297.

See Also

`gaitdnbinomial`, `pospoisson`, `negbinomial`, `zanegbinomial`, `rnbinom`, `CommonVGAMffArguments`, `corbet`, `logff`, `simulate.vlm`, `margeff`.

Examples

```
## Not run:
pdata <- data.frame(x2 = runif(nn <- 1000))
pdata <- transform(pdata,
  y1 = rgaitdnbinom(nn, exp(1), munb.p = exp(0+2*x2), truncate = 0),
  y2 = rgaitdnbinom(nn, exp(3), munb.p = exp(1+2*x2), truncate = 0))
fit <- vglm(cbind(y1, y2) ~ x2, posnegbinomial, pdata, trace = TRUE)
coef(fit, matrix = TRUE)
dim(depvar(fit))  # Using dim(fit@y) is not recommended


# Another artificial data example
pdata2 <- data.frame(munb = exp(2), size = exp(3)); nn <- 1000
pdata2 <- transform(pdata2,
```

```
                    y3 = rgaitdnbinom(nn, size, munb.p = munb,
                                      truncate = 0))
with(pdata2, table(y3))
fit <- vglm(y3 ~ 1, posnegbinomial, data = pdata2, trace = TRUE)
coef(fit, matrix = TRUE)
with(pdata2, mean(y3))  # Sample mean
head(with(pdata2, munb/(1-(size/(size+munb))^size)), 1)  # Popn mean
head(fitted(fit), 3)
head(predict(fit), 3)


# Example: Corbet (1943) butterfly Malaya data
fit <- vglm(ofreq ~ 1, posnegbinomial, weights = species, corbet)
coef(fit, matrix = TRUE)
Coef(fit)
(khat <- Coef(fit)["size"])
pdf2 <- dgaitdnbinom(with(corbet, ofreq), khat,
                     munb.p = fitted(fit), truncate = 0)
print(with(corbet,
           cbind(ofreq, species, fitted = pdf2*sum(species))), dig = 1)
with(corbet,
matplot(ofreq, cbind(species, fitted = pdf2*sum(species)), las = 1,
   xlab = "Observed frequency (of individual butterflies)",
   type = "b", ylab = "Number of species", col = c("blue", "orange"),
   main = "blue 1s = observe; orange 2s = fitted"))

# Data courtesy of Maxim Gerashchenko causes posbinomial() to fail
pnbd.fail <- data.frame(
 y1 = c(1:16, 18:21, 23:28, 33:38, 42, 44, 49:51, 55, 56, 58,
 59, 61:63, 66, 73, 76, 94, 107, 112, 124, 190, 191, 244),
 ofreq = c(130, 80, 38, 23, 22, 11, 21, 14, 6, 7, 9, 9, 9, 4, 4, 5, 1,
           4, 6, 1, 3, 2, 4, 3, 4, 5, 3, 1, 2, 1, 1, 4, 1, 2, 2, 1, 3,
           1, 1, 2, 2, 2, 1, 3, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1))
fit.fail <- vglm(y1 ~ 1, weights = ofreq, posnegbinomial,
               trace = TRUE, data = pnbd.fail)

## End(Not run)
```

Posnorm *The Positive-Normal Distribution*

Description

Density, distribution function, quantile function and random generation for the univariate positive-normal distribution.

Usage

```
dposnorm(x, mean = 0, sd = 1, log = FALSE)
pposnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
```

```
qposnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
rposnorm(n, mean = 0, sd = 1)
```

Arguments

`x, q` vector of quantiles.

`p` vector of probabilities.

`n` number of observations. If `length(n)` > 1 then the length is taken to be the number required.

`mean, sd, log, lower.tail, log.p`
see `rnorm`.

Details

See `posnormal`, the **VGAM** family function for estimating the parameters, for the formula of the probability density function and other details.

Value

`dposnorm` gives the density, `pposnorm` gives the distribution function, `qposnorm` gives the quantile function, and `rposnorm` generates random deviates.

Author(s)

T. W. Yee

See Also

`posnormal`.

Examples

```
## Not run:  m <-  0.8; x <- seq(-1, 4, len = 501)
plot(x, dposnorm(x, m = m), type = "l", las = 1, ylim = 0:1,
     ylab = paste("posnorm(m = ", m, ", sd = 1)"), col = "blue",
     main = "Blue is density, orange is the CDF",
     sub = "Purple lines are the 10,20,...,90 percentiles")
abline(h = 0, col = "grey")
lines(x, pposnorm(x, m = m), col = "orange", type = "l")
probs <- seq(0.1, 0.9, by = 0.1)
Q <- qposnorm(probs, m = m)
lines(Q, dposnorm(Q, m = m), col = "purple", lty = 3, type = "h")
lines(Q, pposnorm(Q, m = m), col = "purple", lty = 3, type = "h")
abline(h = probs, col = "purple", lty = 3)
max(abs(pposnorm(Q, m = m) - probs))  # Should be 0

## End(Not run)
```

posnormal *Positive Normal Distribution Family Function*

Description

Fits a positive (univariate) normal distribution.

Usage

```
posnormal(lmean = "identitylink", lsd = "loglink",
          eq.mean = FALSE, eq.sd = FALSE,
          gmean = exp((-5:5)/2), gsd = exp((-1:5)/2),
          imean = NULL, isd = NULL, probs.y = 0.10, imethod = 1,
          nsimEIM = NULL, zero = "sd")
```

Arguments

`lmean, lsd` Link functions for the mean and standard deviation parameters of the usual univariate normal distribution. They are μ and σ respectively. See `Links` for more choices.

`gmean, gsd, imethod` See `CommonVGAMffArguments` for more information. `gmean` and `gsd` currently operate on a multiplicative scale, on the sample mean and the sample standard deviation, respectively.

`imean, isd` Optional initial values for μ and σ. A `NULL` means a value is computed internally. See `CommonVGAMffArguments` for more information.

`eq.mean, eq.sd` See `CommonVGAMffArguments` for more information. The fact that these arguments are supported results in default constraint matrices being a *permutation* of the identity matrix (effectively *trivial* constraints).

`zero, nsimEIM, probs.y` See `CommonVGAMffArguments` for information.

Details

The positive normal distribution is the ordinary normal distribution but with the probability of zero or less being zero. The rest of the probability density function is scaled up. Hence the probability density function can be written

$$f(y) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}(y-\mu)^2/\sigma^2\right) / \left[1 - \Phi(-\mu/\sigma)\right]$$

where $\Phi()$ is the cumulative distribution function of a standard normal (`pnorm`). Equivalently, this is

$$f(y) = \frac{1}{\sigma} \frac{\phi((y-\mu)/\sigma)}{1 - \Phi(-\mu/\sigma)}$$

where $\phi()$ is the probability density function of a standard normal distribution (`dnorm`).

The mean of Y is

$$E(Y) = \mu + \sigma \frac{\phi(-\mu/\sigma)}{1 - \Phi(-\mu/\sigma)}.$$

This family function handles multiple responses.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Warning

It is recommended that trace = TRUE be used to monitor convergence; sometimes the estimated mean is -Inf and the estimated mean standard deviation is Inf, especially when the sample size is small. Under- or over-flow may occur if the data is ill-conditioned.

Note

The response variable for this family function is the same as uninormal except positive values are required. Reasonably good initial values are needed.

The distribution of the reciprocal of a positive normal random variable is known as an alpha distribution.

Author(s)

Thomas W. Yee

See Also

uninormal, tobit.

Examples

```
pdata <- data.frame(Mean = 1.0, SD = exp(1.0))
pdata <- transform(pdata, y = rposnorm(n <- 1000, m = Mean, sd = SD))

## Not run: with(pdata, hist(y, prob = TRUE, border = "blue",
  main = paste("posnorm(m =", Mean[1], ", sd =", round(SD[1], 2),")")))
## End(Not run)
fit <- vglm(y ~ 1, posnormal, data = pdata, trace = TRUE)
coef(fit, matrix = TRUE)
(Cfit <- Coef(fit))
mygrid <- with(pdata, seq(min(y), max(y), len = 200))
## Not run: lines(mygrid, dposnorm(mygrid, Cfit[1], Cfit[2]), col = "red")
```

pospoisson *Positive Poisson Distribution Family Function*

Description

Fits a positive Poisson distribution.

Usage

```
pospoisson(link = "loglink", type.fitted = c("mean", "lambda",
           "prob0"), expected = TRUE, ilambda = NULL, imethod = 1,
           zero = NULL, gt.1 = FALSE)
```

Arguments

`link`	Link function for the usual mean (lambda) parameter of an ordinary Poisson distribution. See `Links` for more choices.
`expected`	Logical. Fisher scoring is used if `expected = TRUE`, else Newton-Raphson.
`ilambda, imethod, zero`	See `CommonVGAMffArguments` for information.
`type.fitted`	See `CommonVGAMffArguments` for details.
`gt.1`	Logical. Enforce `lambda` > `1`? The default is to enforce `lambda` > `0`.

Details

The positive Poisson distribution is the ordinary Poisson distribution but with the probability of zero being zero. Thus the other probabilities are scaled up (i.e., divided by $1 - P[Y = 0]$). The mean, $\lambda/(1 - \exp(-\lambda))$, can be obtained by the extractor function `fitted` applied to the object.

A related distribution is the zero-inflated Poisson, in which the probability $P[Y = 0]$ involves another parameter ϕ. See `zipoisson`.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, `rrvglm` and `vgam`.

Warning

Under- or over-flow may occur if the data is ill-conditioned.

Note

This family function can handle multiple responses.

Yet to be done: a `quasi.pospoisson` which estimates a dispersion parameter.

Author(s)

Thomas W. Yee

References

Coleman, J. S. and James, J. (1961). The equilibrium size distribution of freely-forming groups. *Sociometry*, **24**, 36–45.

See Also

Gaitdpois, gaitdpoisson, posnegbinomial, poissonff, zapoisson, zipoisson, simulate.vlm, otpospoisson, Pospois.

Examples

```
# Data from Coleman and James (1961)
cjdata <- data.frame(y = 1:6, freq = c(1486, 694, 195, 37, 10, 1))
fit <- vglm(y ~ 1, pospoisson, data = cjdata, weights = freq)
Coef(fit)
summary(fit)
fitted(fit)

pdata <- data.frame(x2 = runif(nn <- 1000))  # Artificial data
pdata <- transform(pdata, lambda = exp(1 - 2 * x2))
pdata <- transform(pdata, y1 = rgaitdpois(nn, lambda, truncate = 0))
with(pdata, table(y1))
fit <- vglm(y1 ~ x2, pospoisson, data = pdata, trace = TRUE, crit = "coef")
coef(fit, matrix = TRUE)
```

---

powerlink *Power Link Function*

Description

Computes the power transformation, including its inverse and the first two derivatives.

Usage

```
powerlink(theta, power = 1, inverse = FALSE, deriv = 0,
          short = TRUE, tag = FALSE)
```

Arguments

`theta`	Numeric or character. See below for further details.
`power`	This denotes the power or exponent.
`inverse, deriv, short, tag`	Details at Links.

Details

The power link function raises a parameter by a certain value of `power`. Care is needed because it is very easy to get numerical problems, e.g., if `power=0.5` and `theta` is negative.

Value

For `powerlink` with `deriv = 0`, then `theta` raised to the power of `power`. And if `inverse = TRUE` then `theta` raised to the power of `1/power`.

For `deriv = 1`, then the function returns *d* `theta` / *d* `eta` as a function of `theta` if `inverse = FALSE`, else if `inverse = TRUE` then it returns the reciprocal.

Note

Numerical problems may occur for certain combinations of `theta` and `power`. Consequently this link function should be used with caution.

Author(s)

Thomas W. Yee

See Also

`Links`, `loglink`.

Examples

```
powerlink("a", power = 2, short = FALSE, tag = TRUE)
powerlink(x <- 1:5)
powerlink(x, power = 2)
max(abs(powerlink(powerlink(x, power = 2),
                  power = 2, inverse = TRUE) - x))  # Should be 0
powerlink(x <- (-5):5, power = 0.5)  # Has NAs

# 1/2 = 0.5
pdata <- data.frame(y = rbeta(n = 1000, shape1 = 2^2, shape2 = 3^2))
fit <- vglm(y ~ 1, betaR(lshape1 = powerlink(power = 0.5), i1 = 3,
                         lshape2 = powerlink(power = 0.5), i2 = 7), data = pdata)
t(coef(fit, matrix = TRUE))
Coef(fit)  # Useful for intercept-only models
vcov(fit, untransform = TRUE)
```

`prats` *Pregnant Rats Toxological Experiment Data*

Description

A small toxological experiment data. The subjects are fetuses from two randomized groups of pregnant rats, and they were given a placebo or chemical treatment. The number with birth defects were recorded, as well as each litter size.

Usage

```
data(prats)
```

Format

A data frame with the following variables.

treatment A `0` means control; a `1` means the chemical treatment.

alive, litter.size The number of fetuses alive at 21 days, out of the number of fetuses alive at 4 days (the litter size).

Details

The data concerns a toxological experiment where the subjects are fetuses from two randomized groups of 16 pregnant rats each, and they were given a placebo or chemical treatment. The number with birth defects and the litter size were recorded. Half the rats were fed a control diet during pregnancy and lactation, and the diet of the other half was treated with a chemical. For each litter the number of pups alive at 4 days and the number of pups that survived the 21 day lactation period, were recorded.

Source

Weil, C. S. (1970) Selection of the valid number of sampling units and a consideration of their combination in toxicological studies involving reproduction, teratogenesis or carcinogenesis. *Food and Cosmetics Toxicology*, **8**(2), 177–182.

References

Williams, D. A. (1975). The Analysis of Binary Responses From Toxicological Experiments Involving Reproduction and Teratogenicity. *Biometrics*, **31**(4), 949–952.

See Also

`betabinomial`, `betabinomialff`.

Examples

```
prats
colSums(subset(prats, treatment == 0))
colSums(subset(prats, treatment == 1))
summary(prats)
```

predictqrrvglm *Predict Method for a CQO fit*

Description

Predicted values based on a constrained quadratic ordination (CQO) object.

Usage

```
predictqrrvglm(object, newdata = NULL,
    type = c("link", "response", "latvar", "terms"),
    se.fit = FALSE, deriv = 0, dispersion = NULL,
    extra = object@extra, varI.latvar = FALSE, refResponse = NULL, ...)
```

Arguments

- `object` Object of class inheriting from "qrrvglm".
- `newdata` An optional data frame in which to look for variables with which to predict. If omitted, the fitted linear predictors are used.
- `type, se.fit, dispersion, extra` See predictvglm.
- `deriv` Derivative. Currently only 0 is handled.
- `varI.latvar, refResponse` Arguments passed into Coef.qrrvglm.
- `...` Currently undocumented.

Details

Obtains predictions from a fitted CQO object. Currently there are lots of limitations of this function; it is unfinished.

Value

See predictvglm.

Note

This function is not robust and has not been checked fully.

Author(s)

T. W. Yee

References

Yee, T. W. (2004). A new technique for maximum-likelihood canonical Gaussian ordination. *Ecological Monographs*, **74**, 685–701.

See Also

cqo, calibrate.qrrvglm.

Examples

```
## Not run:  set.seed(1234)
hspider[, 1:6] <- scale(hspider[, 1:6])  # Standardize the X vars
p1 <- cqo(cbind(Alopacce, Alopcune, Alopfabr, Arctlute,
                Arctperi, Auloalbi, Pardlugu, Pardmont,
                Pardnigr, Pardpull, Trocterr, Zoraspin) ~
          WaterCon + BareSand + FallTwig + CoveMoss + CoveHerb + ReflLux,
          poissonff, data = hspider, Crow1positive = FALSE, I.toler = TRUE)
sort(deviance(p1, history = TRUE))  # A history of all the iterations
head(predict(p1))

# The following should be all 0s:
max(abs(predict(p1, newdata = head(hspider)) - head(predict(p1))))
max(abs(predict(p1, newdata = head(hspider), type = "res")-head(fitted(p1))))

## End(Not run)
```

---

predictvglm *Predict Method for a VGLM fit*

Description

Predicted values based on a vector generalized linear model (VGLM) object.

Usage

```
predictvglm(object, newdata = NULL,
            type = c("link", "response", "terms"),
            se.fit = FALSE, deriv = 0, dispersion = NULL,
            untransform = FALSE,
            type.fitted = NULL, percentiles = NULL, ...)
```

Arguments

object	Object of class inheriting from "vlm", e.g., vglm.
newdata	An optional data frame in which to look for variables with which to predict. If omitted, the fitted linear predictors are used.
type	The value of this argument can be abbreviated. The type of prediction required. The default is the first one, meaning on the scale of the linear predictors. This should be a $n \times M$ matrix. The alternative "response" is on the scale of the response variable, and depending on the family function, this may or may not be the mean. Often this is the fitted value, e.g., fitted(vglmObject) (see fittedvlm). Note that the

	response is output from the `@linkinv` slot, where the `eta` argument is the $n \times M$ matrix of linear predictors. The `"terms"` option returns a matrix giving the fitted values of each term in the model formula on the linear predictor scale. The terms have been centered.
`se.fit`	logical: return standard errors?
`deriv`	Non-negative integer. Currently this must be zero. Later, this may be implemented for general values.
`dispersion`	Dispersion parameter. This may be inputted at this stage, but the default is to use the dispersion parameter of the fitted model.
`type.fitted`	Some **VGAM** family functions have an argument by the same name. If so, then one can obtain fitted values by setting `type = "response"` and choosing a value of `type.fitted` from what's available. If `type.fitted = "quantiles"` is available then the `percentiles` argument can be used to specify what quantile values are requested.
`percentiles`	Used only if `type.fitted = "quantiles"` is available and is selected.
`untransform`	Logical. Reverses any parameter link function. This argument only works if `type = "link", se.fit = FALSE, deriv = 0`. Setting `untransform = TRUE` does not work for all **VGAM** family functions; only ones where there is a one-to-one correspondence between a simple link function and a simple parameter might work.
`...`	Arguments passed into `predictvlm`.

Details

Obtains predictions and optionally estimates standard errors of those predictions from a fitted `vglm` object. By default, each row of the matrix returned can be written as η_i^T, comprising of M components or linear predictors. If there are any offsets, these *are* included.

This code implements *smart prediction* (see `smartpred`).

Value

If `se.fit = FALSE`, a vector or matrix of predictions. If `se.fit = TRUE`, a list with components

`fitted.values`	Predictions
`se.fit`	Estimated standard errors
`df`	Degrees of freedom
`sigma`	The square root of the dispersion parameter (but these are being phased out in the package)

Warning

This function may change in the future.

Note

Setting `se.fit = TRUE` and `type = "response"` will generate an error.

The arguments `type.fitted` and `percentiles` are provided in this function to give more convenience than modifying the `extra` slot directly.

Author(s)

Thomas W. Yee

References

Yee, T. W. (2015). *Vector Generalized Linear and Additive Models: With an Implementation in R*. New York, USA: *Springer*.

Yee, T. W. and Hastie, T. J. (2003). Reduced-rank vector generalized linear models. *Statistical Modelling*, **3**, 15–41.

See Also

predict, vglm, predictvlm, smartpred, calibrate.

Examples

```
# Illustrates smart prediction
pneumo <- transform(pneumo, let = log(exposure.time))
fit <- vglm(cbind(normal, mild, severe) ~ poly(c(scale(let)), 2),
            propodds, pneumo, trace = TRUE, x.arg = FALSE)
class(fit)

(q0 <- head(predict(fit)))
(q1 <- predict(fit, newdata = head(pneumo)))
(q2 <- predict(fit, newdata = head(pneumo)))
all.equal(q0, q1)  # Should be TRUE
all.equal(q1, q2)  # Should be TRUE

head(predict(fit))
head(predict(fit, untransform = TRUE))

p0 <- head(predict(fit, type = "response"))
p1 <- head(predict(fit, type = "response", newdata = pneumo))
p2 <- head(predict(fit, type = "response", newdata = pneumo))
p3 <- head(fitted(fit))
all.equal(p0, p1)  # Should be TRUE
all.equal(p1, p2)  # Should be TRUE
all.equal(p2, p3)  # Should be TRUE

predict(fit, type = "terms", se = TRUE)
```

---

prentice74 *Prentice (1974) Log-gamma Distribution*

Description

Estimation of a 3-parameter log-gamma distribution described by Prentice (1974).

Usage

```
prentice74(llocation = "identitylink", lscale = "loglink",
           lshape = "identitylink", ilocation = NULL, iscale = NULL,
           ishape = NULL, imethod = 1,
           glocation.mux = exp((-4:4)/2), gscale.mux = exp((-4:4)/2),
           gshape = qt(ppoints(6), df = 1), probs.y = 0.3,
           zero = c("scale", "shape"))
```

Arguments

`llocation, lscale, lshape`
: Parameter link function applied to the location parameter a, positive scale parameter b and the shape parameter q, respectively. See `Links` for more choices.

`ilocation, iscale`
: Initial value for a and b, respectively. The defaults mean an initial value is determined internally for each.

`ishape`
: Initial value for q. If failure to converge occurs, try some other value. The default means an initial value is determined internally.

`imethod, zero`
: See `CommonVGAMffArguments` for information.

`glocation.mux, gscale.mux, gshape, probs.y`
: See `CommonVGAMffArguments` for information.

Details

The probability density function is given by

$$f(y; a, b, q) = |q| \exp(w/q^2 - e^w)/(b\,\Gamma(1/q^2)),$$

for shape parameter $q \neq 0$, positive scale parameter $b > 0$, location parameter a, and all real y. Here, $w = (y - a)q/b + \psi(1/q^2)$ where ψ is the digamma function, `digamma`. The mean of Y is a (returned as the fitted values). This is a different parameterization compared to `lgamma3`.

Special cases: $q = 0$ is the normal distribution with standard deviation b, $q = -1$ is the extreme value distribution for maximums, $q = 1$ is the extreme value distribution for minima (Weibull). If $q > 0$ then the distribution is left skew, else $q < 0$ is right skew.

Value

An object of class "vglmff" (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Warning

The special case $q = 0$ is not handled, therefore estimates of q too close to zero may cause numerical problems.

Note

The notation used here differs from Prentice (1974): $\alpha = a$, $\sigma = b$. Fisher scoring is used.

Author(s)

T. W. Yee

References

Prentice, R. L. (1974). A log gamma model and its maximum likelihood estimation. *Biometrika*, **61**, 539–544.

See Also

lgamma3, lgamma, gengamma.stacy.

Examples

```
pdata <- data.frame(x2 = runif(nn <- 1000))
pdata <- transform(pdata, loc = -1 + 2*x2, Scale = exp(1))
pdata <- transform(pdata, y = rlgamma(nn, loc = loc, scale = Scale, shape = 1))
fit <- vglm(y ~ x2, prentice74(zero = 2:3), data = pdata, trace = TRUE)
coef(fit, matrix = TRUE)  # Note the coefficients for location
```

---

prinia *Yellow-bellied Prinia*

Description

A data frame with yellow-bellied Prinia.

Usage

```
data(prinia)
```

Format

A data frame with 151 observations on the following 23 variables.

length a numeric vector, the scaled wing length (zero mean and unit variance).

fat a numeric vector, fat index; originally 1 (no fat) to 4 (very fat) but converted to 0 (no fat) versus 1 otherwise.

cap a numeric vector, number of times the bird was captured or recaptured.

noncap a numeric vector, number of times the bird was not captured.

y01, y02, y03, y04, y05, y06 a numeric vector of 0s and 1s; for noncapture and capture resp.

y07, y08, y09, y10, y11, y12 same as above.

y13, y14, y15, y16, y17, y18, y19 same as above.

Details

The yellow-bellied Prinia *Prinia flaviventris* is a common bird species located in Southeast Asia. A capture–recapture experiment was conducted at the Mai Po Nature Reserve in Hong Kong during 1991, where captured individuals had their wing lengths measured and fat index recorded. A total of 19 weekly capture occasions were considered, where 151 distinct birds were captured.

More generally, the prinias are a genus of small insectivorous birds, and are sometimes referred to as *wren-warblers*. They are a little-known group of the tropical and subtropical Old World, the roughly 30 species being divided fairly equally between Africa and Asia.

Source

Thanks to Paul Yip for permission to make this data available.

Hwang, W.-H. and Huggins, R. M. (2007) Application of semiparametric regression models in the analysis of capture–recapture experiments. *Australian and New Zealand Journal of Statistics* **49**, 191–202.

Examples

```
head(prinia)
summary(prinia)
rowSums(prinia[, c("cap", "noncap")])  # 19s

#  Fit a positive-binomial distribution (M.h) to the data:
fit1 <- vglm(cbind(cap, noncap) ~ length + fat, posbinomial, prinia)

#  Fit another positive-binomial distribution (M.h) to the data:
#  The response input is suitable for posbernoulli.*-type functions.
fit2 <- vglm(cbind(y01, y02, y03, y04, y05, y06, y07, y08, y09, y10,
                   y11, y12, y13, y14, y15, y16, y17, y18, y19) ~
             length + fat, posbernoulli.b(drop.b = FALSE ~ 0), prinia)
```

---

probitlink *Probit Link Function*

Description

Computes the probit transformation, including its inverse and the first two derivatives.

Usage

```
probitlink(theta, bvalue = NULL, inverse = FALSE, deriv = 0,
           short = TRUE, tag = FALSE)
```

Arguments

`theta`	Numeric or character. See below for further details.
`bvalue`	See `Links`.
`inverse, deriv, short, tag`	Details at `Links`.

Details

The probit link function is commonly used for parameters that lie in the unit interval. It is the inverse CDF of the standard normal distribution. Numerical values of `theta` close to 0 or 1 or out of range result in `Inf`, `-Inf`, `NA` or `NaN`.

Value

For `deriv = 0`, the probit of `theta`, i.e., `qnorm(theta)` when `inverse = FALSE`, and if `inverse = TRUE` then `pnorm(theta)`.

For `deriv = 1`, then the function returns *d* `eta` / *d* `theta` as a function of `theta` if `inverse = FALSE`, else if `inverse = TRUE` then it returns the reciprocal.

Note

Numerical instability may occur when `theta` is close to 1 or 0. One way of overcoming this is to use `bvalue`.

In terms of the threshold approach with cumulative probabilities for an ordinal response this link function corresponds to the univariate normal distribution (see `uninormal`).

Author(s)

Thomas W. Yee

References

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*, 2nd ed. London: Chapman & Hall.

See Also

`Links`, `logitlink`, `clogloglink`, `cauchitlink`, `Normal`.

Examples

```
p <- seq(0.01, 0.99, by = 0.01)
probitlink(p)
max(abs(probitlink(probitlink(p), inverse = TRUE) - p))  # Should be 0

p <- c(seq(-0.02, 0.02, by = 0.01), seq(0.97, 1.02, by = 0.01))
probitlink(p)  # Has NAs
probitlink(p, bvalue = .Machine$double.eps)  # Has no NAs

## Not run: p <- seq(0.01, 0.99, by = 0.01); par(lwd = (mylwd <- 2))
plot(p, logitlink(p), type = "l", col = "limegreen", ylab = "transformation",
     las = 1, main = "Some probability link functions")
lines(p,  probitlink(p), col = "purple")
lines(p, clogloglink(p), col = "chocolate")
lines(p, cauchitlink(p), col = "tan")
abline(v = 0.5, h = 0, lty = "dashed")
legend(0.1, 4, c("logitlink", "probitlink", "clogloglink", "cauchitlink"),
```

```
        col = c("limegreen", "purple", "chocolate", "tan"), lwd = mylwd)
par(lwd = 1)
## End(Not run)
```

---

profilevglm *Method for Profiling vglm Objects*

Description

Investigates the profile log-likelihood function for a fitted model of class "vglm".

Usage

```
profilevglm(object, which = 1:p.vlm, alpha = 0.01,
            maxsteps = 10, del = zmax/5, trace = NULL, ...)
```

Arguments

object	the original fitted model object.
which	the original model parameters which should be profiled. This can be a numeric or character vector. By default, all parameters are profiled.
alpha	highest significance level allowed for the profiling.
maxsteps	maximum number of points to be used for profiling each parameter.
del	suggested change on the scale of the profile t-statistics. Default value chosen to allow profiling at about 10 parameter values.
trace	logical: should the progress of profiling be reported? The default is to use the trace value from the fitted object; see vglm.control for details.
...	further arguments passed to or from other methods.

Details

This function is called by confintvglm to do the profiling. See also profile.glm for details.

Value

A list of classes "profile.glm" and "profile" with an element for each parameter being profiled. The elements are data-frames with two variables

par.vals	a matrix of parameter values for each fitted model.
tau	the profile t-statistics.

Author(s)

T. W. Yee adapted this function from profile.glm, written originally by D. M. Bates and W. N. Venables. (For S in 1996.) The help file was also used as a template.

See Also

vglm, confintvglm, lrt.stat, profile, profile.glm, plot.profile in **MASS** or **stats**.

Examples

```
pneumo <- transform(pneumo, let = log(exposure.time))
fit1 <- vglm(cbind(normal, mild, severe) ~ let, propodds,
             trace = TRUE, data = pneumo)
pfit1 <- profile(fit1, trace = FALSE)
confint(fit1, method = "profile", trace = FALSE)
```

---

propodds *Proportional Odds Model for Ordinal Regression*

Description

Fits the proportional odds model to a (preferably ordered) factor response.

Usage

```
propodds(reverse = TRUE, whitespace = FALSE, ynames = FALSE,
   Thresh = NULL, Trev = reverse, Tref = if (Trev) "M" else 1)
```

Arguments

`reverse, whitespace`
: Logical. Fed into arguments of the same name in cumulative.

`ynames`
: See multinomial for information.

`Thresh, Trev, Tref`
: Fed into arguments of the same name in cumulative.

Details

The *proportional odds model* is a special case from the class of *cumulative link models*. It involves a logit link applied to cumulative probabilities and a strong *parallelism* assumption. A parallelism assumption means there is less chance of numerical problems because the fitted probabilities will remain between 0 and 1; however the *parallelism* assumption ought to be checked, e.g., via a likelihood ratio test. This **VGAM** family function is merely a shortcut for `cumulative(reverse = reverse, link = "logit", parallel = TRUE)`. Please see cumulative for more details on this model.

Value

An object of class `"vglmff"` (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Warning

No check is made to verify that the response is ordinal if the response is a matrix; see ordered.

Author(s)

Thomas W. Yee

References

See cumulative.

See Also

cumulative, R2latvar.

Examples

```
# Fit the proportional odds model, McCullagh and Nelder (1989,p.179)
pneumo <- transform(pneumo, let = log(exposure.time))
(fit <- vglm(cbind(normal, mild, severe) ~ let, propodds, pneumo))
depvar(fit)  # Sample proportions
weights(fit, type = "prior")  # Number of observations
coef(fit, matrix = TRUE)
constraints(fit)  # Constraint matrices
summary(fit)

# Check that the model is linear in let ----------------------
fit2 <- vgam(cbind(normal, mild, severe) ~ s(let, df = 2), propodds,
             pneumo)
## Not run:  plot(fit2, se = TRUE, lcol = 2, scol = 2)

# Check the proportional odds assumption with a LRT ----------
(fit3 <- vglm(cbind(normal, mild, severe) ~ let,
              cumulative(parallel = FALSE, reverse = TRUE), pneumo))
pchisq(deviance(fit) - deviance(fit3),
       df = df.residual(fit) - df.residual(fit3), lower.tail = FALSE)
lrtest(fit3, fit)  # Easier
```

---

prplot *Probability Plots for Categorical Data Analysis*

Description

Plots the fitted probabilities for some very simplified special cases of categorical data analysis models.

Usage

```
prplot(object, control = prplot.control(...), ...)

prplot.control(xlab = NULL, ylab = "Probability", main = NULL, xlim = NULL,
    ylim = NULL, lty = par()$lty, col = par()$col, rcol = par()$col,
    lwd = par()$lwd, rlwd = par()$lwd, las = par()$las, rug.arg = FALSE, ...)
```

Arguments

`object` Currently only an `cumulative` object. This includes a `propodds` object since that **VGAM** family function is a special case of `cumulative`.

`control` List containing some basic graphical parameters.

`xlab, ylab, main, xlim, ylim, lty` See `par` and `...` below.

`col, rcol, lwd, rlwd, las, rug.arg` See `par` and `...` below. Arguments starting with `r` refer to the rug. Argument `rug.arg` is logical: add a rug for the distinct values of the explanatory variable?

`...` Arguments such as `xlab` which are fed into `prplot.control()`. Only a small selection of graphical arguments from `par` are offered.

Details

For models involving one term in the RHS of the formula this function plots the fitted probabilities against the single explanatory variable.

Value

The object is returned invisibly with the `preplot` slot assigned. This is obtained by a call to `plotvgam()`.

Note

This function is rudimentary.

See Also

`cumulative`.

Examples

```
pneumo <- transform(pneumo, let = log(exposure.time))
fit <- vglm(cbind(normal, mild, severe) ~ let, propodds, data = pneumo)
M <- npred(fit)  # Or fit@misc$M
## Not run:  prplot(fit)
prplot(fit, lty = 1:M, col = (1:M)+2, rug = TRUE, las = 1,
       ylim = c(0, 1), rlwd = 2)
## End(Not run)
```

put.smart *Adds a List to the End of the List ".smart.prediction"*

Description

Adds a list to the end of the list `.smart.prediction` in `smartpredenv`.

Usage

```
put.smart(smart)
```

Arguments

`smart`	a list containing parameters needed later for smart prediction.

Details

`put.smart` is used in `"write"` mode within a smart function. It saves parameters at the time of model fitting, which are later used for prediction. The function `put.smart` is the opposite of `get.smart`, and both deal with the same contents.

Value

Nothing is returned.

Side Effects

The variable `.smart.prediction.counter` in `smartpredenv` is incremented beforehand, and `.smart.prediction[[.smar` is assigned the list `smart`. If the list `.smart.prediction` in `smartpredenv` is not long enough to hold `smart`, then it is made larger, and the variable `.max.smart` in `smartpredenv` is adjusted accordingly.

See Also

`get.smart`.

Examples

```
print(sm.min1)
```

qrrvglm.control *Control Function for QRR-VGLMs (CQO)*

Description

Algorithmic constants and parameters for a constrained quadratic ordination (CQO), by fitting a *quadratic reduced-rank vector generalized linear model* (QRR-VGLM), are set using this function. It is the control function for cqo.

Usage

```
qrrvglm.control(Rank = 1, Bestof = if (length(Cinit)) 1 else 10,
    checkwz = TRUE, Cinit = NULL, Crow1positive = TRUE,
    epsilon = 1.0e-06, EqualTolerances = NULL, eq.tolerances = TRUE,
    Etamat.colmax = 10, FastAlgorithm = TRUE, GradientFunction = TRUE,
    Hstep = 0.001, isd.latvar = rep_len(c(2, 1, rep_len(0.5, Rank)),
    Rank), iKvector = 0.1, iShape = 0.1, ITolerances = NULL,
    I.tolerances = FALSE, maxitl = 40, imethod = 1,
    Maxit.optim = 250, MUXfactor = rep_len(7, Rank),
    noRRR = ~ 1, Norrr = NA, optim.maxit = 20,
    Parscale = if (I.tolerances) 0.001 else 1.0,
    sd.Cinit = 0.02, SmallNo = 5.0e-13, trace = TRUE,
    Use.Init.Poisson.QO = TRUE,
    wzepsilon = .Machine$double.eps^0.75, ...)
```

Arguments

In the following, R is the Rank, M is the number of linear predictors, and S is the number of responses (species). Thus $M = S$ for binomial and Poisson responses, and $M = 2S$ for the negative binomial and 2-parameter gamma distributions.

Rank
: The numerical rank R of the model, i.e., the number of ordination axes. Must be an element from the set $\{1,2,\ldots,\min(M,p_2)\}$ where the vector of explanatory variables x is partitioned into (x_1,x_2), which is of dimension $p_1 + p_2$. The variables making up x_1 are given by the terms in the noRRR argument, and the rest of the terms comprise x_2.

Bestof
: Integer. The best of Bestof models fitted is returned. This argument helps guard against local solutions by (hopefully) finding the global solution from many fits. The argument has value 1 if an initial value for C is inputted using Cinit.

checkwz
: logical indicating whether the diagonal elements of the working weight matrices should be checked whether they are sufficiently positive, i.e., greater than wzepsilon. If not, any values less than wzepsilon are replaced with this value.

Cinit
: Optional initial C matrix, which must be a p_2 by R matrix. The default is to apply .Init.Poisson.QO() to obtain initial values.

`Crow1positive` Logical vector of length `Rank` (recycled if necessary): are the elements of the first row of C positive? For example, if `Rank` is 4, then specifying `Crow1positive = c(FALSE, TRUE)` will force $C[1,1]$ and $C[1,3]$ to be negative, and $C[1,2]$ and $C[1,4]$ to be positive. This argument allows for a reflection in the ordination axes because the coefficients of the latent variables are unique up to a sign.

`epsilon` Positive numeric. Used to test for convergence for GLMs fitted in C. Larger values mean a loosening of the convergence criterion. If an error code of 3 is reported, try increasing this value.

`eq.tolerances` Logical indicating whether each (quadratic) predictor will have equal tolerances. Having `eq.tolerances = TRUE` can help avoid numerical problems, especially with binary data. Note that the estimated (common) tolerance matrix may or may not be positive-definite. If it is then it can be scaled to the R by R identity matrix, i.e., made equivalent to `I.tolerances = TRUE`. Setting `I.tolerances = TRUE` will *force* a common R by R identity matrix as the tolerance matrix to the data even if it is not appropriate. In general, setting `I.tolerances = TRUE` is preferred over `eq.tolerances = TRUE` because, if it works, it is much faster and uses less memory. However, `I.tolerances = TRUE` requires the environmental variables to be scaled appropriately. See **Details** for more details.

`EqualTolerances`
Defunct argument. Use `eq.tolerances` instead.

`Etamat.colmax` Positive integer, no smaller than `Rank`. Controls the amount of memory used by `.Init.Poisson.QO()`. It is the maximum number of columns allowed for the pseudo-response and its weights. In general, the larger the value, the better the initial value. Used only if `Use.Init.Poisson.QO = TRUE`.

`FastAlgorithm` Logical. Whether a new fast algorithm is to be used. The fast algorithm results in a large speed increases compared to Yee (2004). Some details of the fast algorithm are found in Appendix A of Yee (2006). Setting `FastAlgorithm = FALSE` will give an error.

`GradientFunction`
Logical. Whether `optim`'s argument `gr` is used or not, i.e., to compute gradient values. Used only if `FastAlgorithm` is `TRUE`. The default value is usually faster on most problems.

`Hstep` Positive value. Used as the step size in the finite difference approximation to the derivatives by `optim`.

`isd.latvar` Initial standard deviations for the latent variables (site scores). Numeric, positive and of length R (recycled if necessary). This argument is used only if `I.tolerances = TRUE`. Used by `.Init.Poisson.QO()` to obtain initial values for the constrained coefficients C adjusted to a reasonable value. It adjusts the spread of the site scores relative to a common species tolerance of 1 for each ordination axis. A value between 0.5 and 10 is recommended; a value such as 10 means that the range of the environmental space is very large relative to the niche width of the species. The successive values should decrease because the first ordination axis should have the most spread of site scores, followed by the second ordination axis, etc.

`iKvector, iShape`
Numeric, recycled to length S if necessary. Initial values used for estimating the positive k and λ parameters of the negative binomial and 2-parameter

gamma distributions respectively. For further information see negbinomial and gamma2. These arguments override the ik and ishape arguments in negbinomial and gamma2.

I.tolerances Logical. If TRUE then the (common) tolerance matrix is the R by R identity matrix by definition. Note that having I.tolerances = TRUE implies eq.tolerances = TRUE, but not vice versa. Internally, the quadratic terms will be treated as offsets (in GLM jargon) and so the models can potentially be fitted very efficiently. *However, it is a very good idea to center and scale all numerical variables in the x_2 vector.* See **Details** for more details. The success of I.tolerances = TRUE often depends on suitable values for isd.latvar and/or MUXfactor.

ITolerances Defunct argument. Use I.tolerances instead.

maxitl Maximum number of times the optimizer is called or restarted. Most users should ignore this argument.

imethod Method of initialization. A positive integer 1 or 2 or 3 etc. depending on the **VGAM** family function. Currently it is used for negbinomial and gamma2 only, and used within the C.

Maxit.optim Positive integer. Number of iterations given to the function optim at each of the optim.maxit iterations.

MUXfactor Multiplication factor for detecting large offset values. Numeric, positive and of length R (recycled if necessary). This argument is used only if I.tolerances = TRUE. Offsets are -0.5 multiplied by the sum of the squares of all R latent variable values. If the latent variable values are too large then this will result in numerical problems. By too large, it is meant that the standard deviation of the latent variable values are greater than MUXfactor[r] * isd.latvar[r] for r=1:Rank (this is why centering and scaling all the numerical predictor variables in x_2 is recommended). A value about 3 or 4 is recommended. If failure to converge occurs, try a slightly lower value.

optim.maxit Positive integer. Number of times optim is invoked. At iteration i, the ith value of Maxit.optim is fed into optim.

noRRR Formula giving terms that are *not* to be included in the reduced-rank regression (or formation of the latent variables), i.e., those belong to x_1. Those variables which do not make up the latent variable (reduced-rank regression) correspond to the B_1 matrix. The default is to omit the intercept term from the latent variables.

Norrr Defunct. Please use noRRR. Use of Norrr will become an error soon.

Parscale Numerical and positive-valued vector of length C (recycled if necessary). Passed into optim(..., control = list(parscale = Parscale)); the elements of C become C / Parscale. Setting I.tolerances = TRUE results in line searches that are very large, therefore C has to be scaled accordingly to avoid large step sizes. See **Details** for more information. It's probably best to leave this argument alone.

sd.Cinit Standard deviation of the initial values for the elements of C. These are normally distributed with mean zero. This argument is used only if Use.Init.Poisson.QO = FALSE and C is not inputted using Cinit.

`trace` Logical indicating if output should be produced for each iteration. The default is `TRUE` because the calculations are numerically intensive, meaning it may take a long time, so that the user might think the computer has locked up if `trace = FALSE`.

`SmallNo` Positive numeric between `.Machine$double.eps` and `0.0001`. Used to avoid under- or over-flow in the IRLS algorithm. Used only if `FastAlgorithm` is `TRUE`.

`Use.Init.Poisson.QO` Logical. If `TRUE` then the function `.Init.Poisson.QO()` is used to obtain initial values for the canonical coefficients C. If `FALSE` then random numbers are used instead.

`wzepsilon` Small positive number used to test whether the diagonals of the working weight matrices are sufficiently positive.

`...` Ignored at present.

Details

Recall that the central formula for CQO is

$$\eta = B_1^T x_1 + A\nu + \sum_{m=1}^{M} (\nu^T D_m \nu) e_m$$

where x_1 is a vector (usually just a 1 for an intercept), x_2 is a vector of environmental variables, $\nu = C^T x_2$ is a R-vector of latent variables, e_m is a vector of 0s but with a 1 in the mth position. QRR-VGLMs are an extension of RR-VGLMs and allow for maximum likelihood solutions to constrained quadratic ordination (CQO) models.

Having `I.tolerances = TRUE` means all the tolerance matrices are the order-R identity matrix, i.e., it *forces* bell-shaped curves/surfaces on all species. This results in a more difficult optimization problem (especially for 2-parameter models such as the negative binomial and gamma) because of overflow errors and it appears there are more local solutions. To help avoid the overflow errors, scaling C by the factor `Parscale` can help enormously. Even better, scaling C by specifying `isd.latvar` is more understandable to humans. If failure to converge occurs, try adjusting `Parscale`, or better, setting `eq.tolerances = TRUE` (and hope that the estimated tolerance matrix is positive-definite). To fit an equal-tolerances model, it is firstly best to try setting `I.tolerances = TRUE` and varying `isd.latvar` and/or `MUXfactor` if it fails to converge. If it still fails to converge after many attempts, try setting `eq.tolerances = TRUE`, however this will usually be a lot slower because it requires a lot more memory.

With a $R > 1$ model, the latent variables are always uncorrelated, i.e., the variance-covariance matrix of the site scores is a diagonal matrix.

If setting `eq.tolerances = TRUE` is used and the common estimated tolerance matrix is positive-definite then that model is effectively the same as the `I.tolerances = TRUE` model (the two are transformations of each other). In general, `I.tolerances = TRUE` is numerically more unstable and presents a more difficult problem to optimize; the arguments `isd.latvar` and/or `MUXfactor` often must be assigned some good value(s) (possibly found by trial and error) in order for convergence to occur. Setting `I.tolerances = TRUE` *forces* a bell-shaped curve or surface onto all the species data, therefore this option should be used with deliberation. If unsuitable, the resulting fit may be very misleading. Usually it is a good idea for the user to set `eq.tolerances = FALSE` to see which species appear to have a bell-shaped curve or surface. Improvements to the fit can often be achieved using transformations, e.g., nitrogen concentration to log nitrogen concentration.

Fitting a CAO model (see `cao`) first is a good idea for pre-examining the data and checking whether it is appropriate to fit a CQO model.

Value

A list with components matching the input names.

Warning

The default value of `Bestof` is a bare minimum for many datasets, therefore it will be necessary to increase its value to increase the chances of obtaining the global solution.

Note

When `I.tolerances = TRUE` it is a good idea to apply `scale` to all the numerical variables that make up the latent variable, i.e., those of x_2. This is to make them have mean 0, and hence avoid large offset values which cause numerical problems.

This function has many arguments that are common with `rrvglm.control` and `vglm.control`.

It is usually a good idea to try fitting a model with `I.tolerances = TRUE` first, and if convergence is unsuccessful, then try `eq.tolerances = TRUE` and `I.tolerances = FALSE`. Ordination diagrams with `eq.tolerances = TRUE` have a natural interpretation, but with `eq.tolerances = FALSE` they are more complicated and requires, e.g., contours to be overlaid on the ordination diagram (see `lvplot.qrrvglm`).

In the example below, an equal-tolerances CQO model is fitted to the hunting spiders data. Because `I.tolerances = TRUE`, it is a good idea to center all the x_2 variables first. Upon fitting the model, the actual standard deviation of the site scores are computed. Ideally, the `isd.latvar` argument should have had this value for the best chances of getting good initial values. For comparison, the model is refitted with that value and it should run more faster and reliably.

Author(s)

Thomas W. Yee

References

Yee, T. W. (2004). A new technique for maximum-likelihood canonical Gaussian ordination. *Ecological Monographs*, **74**, 685–701.

Yee, T. W. (2006). Constrained additive ordination. *Ecology*, **87**, 203–213.

See Also

`cqo`, `rcqo`, `Coef.qrrvglm`, `Coef.qrrvglm-class`, `optim`, `binomialff`, `poissonff`, `negbinomial`, `gamma2`.

Examples

```
## Not run:  # Poisson CQO with equal tolerances
set.seed(111)  # This leads to the global solution
hspider[,1:6] <- scale(hspider[,1:6])  # Good when I.tolerances = TRUE
p1 <- cqo(cbind(Alopacce, Alopcune, Alopfabr,
                Arctlute, Arctperi, Auloalbi,
                Pardlugu, Pardmont, Pardnigr,
                Pardpull, Trocterr, Zoraspin) ~
          WaterCon + BareSand + FallTwig +
          CoveMoss + CoveHerb + ReflLux,
          poissonff, data = hspider, eq.tolerances = TRUE)
sort(deviance(p1, history = TRUE))  # Iteration history

(isd.latvar <- apply(latvar(p1), 2, sd))  # Approx isd.latvar

# Refit the model with better initial values
set.seed(111)  # This leads to the global solution
p1 <- cqo(cbind(Alopacce, Alopcune, Alopfabr,
                Arctlute, Arctperi, Auloalbi,
                Pardlugu, Pardmont, Pardnigr,
                Pardpull, Trocterr, Zoraspin) ~
          WaterCon + BareSand + FallTwig +
          CoveMoss + CoveHerb + ReflLux,
          I.tolerances = TRUE, poissonff, data = hspider,
          isd.latvar = isd.latvar)  # Note this
sort(deviance(p1, history = TRUE))  # Iteration history

## End(Not run)
```

---

qtplot.gumbel *Quantile Plot for Gumbel Regression*

Description

Plots quantiles associated with a Gumbel model.

Usage

```
qtplot.gumbel(object, show.plot = TRUE,
    y.arg = TRUE, spline.fit = FALSE, label = TRUE,
    R = object@misc$R, percentiles = object@misc$percentiles,
    add.arg = FALSE, mpv = object@misc$mpv,
    xlab = NULL, ylab = "", main = "",
    pch = par()$pch, pcol.arg = par()$col,
    llty.arg = par()$lty, lcol.arg = par()$col, llwd.arg = par()$lwd,
    tcol.arg = par()$col, tadj = 1, ...)
```

Arguments

object	A **VGAM** extremes model of the Gumbel type, produced by modelling functions such as vglm and vgam, and with a family function that is either gumbel or gumbelff.
show.plot	Logical. Plot it? If FALSE no plot will be done.
y.arg	Logical. Add the raw data on to the plot?
spline.fit	Logical. Use a spline fit through the fitted percentiles? This can be useful if there are large gaps between some values along the covariate.
label	Logical. Label the percentiles?
R	See gumbel.
percentiles	See gumbel.
add.arg	Logical. Add the plot to an existing plot?
mpv	See gumbel.
xlab	Caption for the x-axis. See par.
ylab	Caption for the y-axis. See par.
main	Title of the plot. See title.
pch	Plotting character. See par.
pcol.arg	Color of the points. See the col argument of par.
llty.arg	Line type. Line type. See the lty argument of par.
lcol.arg	Color of the lines. See the col argument of par.
llwd.arg	Line width. See the lwd argument of par.
tcol.arg	Color of the text (if label is TRUE). See the col argument of par.
tadj	Text justification. See the adj argument of par.
...	Arguments passed into the plot function when setting up the entire plot. Useful arguments here include sub and las.

Details

There should be a single covariate such as time. The quantiles specified by percentiles are plotted.

Value

The object with a list called qtplot in the post slot of object. (If show.plot = FALSE then just the list is returned.) The list contains components

fitted.values	The percentiles of the response, possibly including the MPV.
percentiles	The percentiles (small vector of values between 0 and 100.

Note

Unlike gumbel, one cannot have percentiles = NULL.

Author(s)

Thomas W. Yee

See Also

gumbel.

Examples

```
ymat <- as.matrix(venice[, paste("r", 1:10, sep = "")])
fit1 <- vgam(ymat ~ s(year, df = 3), gumbel(R = 365, mpv = TRUE),
             data = venice, trace = TRUE, na.action = na.pass)
head(fitted(fit1))

## Not run:  par(mfrow = c(1, 1), bty = "l", xpd = TRUE, las = 1)
qtplot(fit1, mpv = TRUE, lcol = c(1, 2, 5), tcol = c(1, 2, 5),
       lwd = 2, pcol = "blue", tadj = 0.4, ylab = "Sea level (cm)")

qtplot(fit1, perc = 97, mpv = FALSE, lcol = 3, tcol = 3,
       lwd = 2, tadj = 0.4, add = TRUE) -> saved
head(saved@post$qtplot$fitted)

## End(Not run)
```

---

qtplot.lmscreg   *Quantile Plot for LMS Quantile Regression*

Description

Plots quantiles associated with a LMS quantile regression.

Usage

```
qtplot.lmscreg(object, newdata = NULL,
               percentiles = object@misc$percentiles,
               show.plot = TRUE, ...)
```

Arguments

object	A **VGAM** quantile regression model, i.e., an object produced by modelling functions such as vglm and vgam with a family function beginning with "lms.", e.g., lms.yjn.
newdata	Optional data frame for computing the quantiles. If missing, the original data is used.
percentiles	Numerical vector with values between 0 and 100 that specify the percentiles (quantiles). The default are the percentiles used when the model was fitted.
show.plot	Logical. Plot it? If FALSE no plot will be done.
...	Graphical parameter that are passed into plotqtplot.lmscreg.

Details

The 'primary' variable is defined as the main covariate upon which the regression or smoothing is performed. For example, in medical studies, it is often the age. In **VGAM**, it is possible to handle more than one covariate, however, the primary variable must be the first term after the intercept.

Value

A list with the following components.

`fitted.values` A vector of fitted percentile values.

`percentiles` The percentiles used.

Note

`plotqtplot.lmscreg` does the actual plotting.

Author(s)

Thomas W. Yee

References

Yee, T. W. (2004). Quantile regression via vector generalized additive models. *Statistics in Medicine*, **23**, 2295–2315.

See Also

`plotqtplot.lmscreg`, `deplot.lmscreg`, `lms.bcn`, `lms.bcg`, `lms.yjn`.

Examples

```
## Not run:
fit <- vgam(BMI ~ s(age, df = c(4, 2)), lms.bcn(zero=1), bmi.nz)
qtplot(fit)
qtplot(fit, perc = c(25, 50, 75, 95), lcol = 4, tcol = 4, llwd = 2)

## End(Not run)
```

Qvar *Quasi-variances Preprocessing Function*

Description

Takes a `vglm` fit or a variance-covariance matrix, and preprocesses it for `rcim` and `uninormal` so that quasi-variances can be computed.

Usage

```
Qvar(object, factorname = NULL, which.linpred = 1,
     coef.indices = NULL, labels = NULL,
     dispersion = NULL, reference.name = "(reference)", estimates = NULL)
```

Arguments

`object`	A "vglm" object or a variance-covariance matrix, e.g., `vcov(vglm.object)`. The former is preferred since it contains all the information needed. If a matrix then `factorname` and/or `coef.indices` should be specified to identify the factor.
`which.linpred`	A single integer from the set `1:M`. Specifies which linear predictor to use. Let the value of `which.linpred` be called j. Then the factor should appear in that linear predictor, hence the jth row of the constraint matrix corresponding to the factor should have at least one nonzero value. Currently the jth row must have exactly one nonzero value because programming it for more than one nonzero value is difficult.
`factorname`	Character. If the `vglm` object contains more than one factor as explanatory variable then this argument should be the name of the factor of interest. If `object` is a variance-covariance matrix then this argument should also be specified.
`labels`	Character. Optional, for labelling the variance-covariance matrix.
`dispersion`	Numeric. Optional, passed into `vcov()` with the same argument name.
`reference.name`	Character. Label for for the reference level.
`coef.indices`	Optional numeric vector of length at least 3 specifying the indices of the factor from the variance-covariance matrix.
`estimates`	an optional vector of estimated coefficients (redundant if `object` is a model).

Details

Suppose a factor with L levels is an explanatory variable in a regression model. By default, R treats the first level as baseline so that its coefficient is set to zero. It estimates the other $L - 1$ coefficients, and with its associated standard errors, this is the conventional output. From the complete variance-covariance matrix one can compute L quasi-variances based on all pairwise difference of the coefficients. They are based on an approximation, and can be treated as uncorrelated. In minimizing the relative (not absolute) errors it is not hard to see that the estimation involves a RCIM (`rcim`) with an exponential link function (`explink`).

If `object` is a model, then at least one of `factorname` or `coef.indices` must be non-`NULL`. The value of `coef.indices`, if non-`NULL`, determines which rows and columns of the model's variance-covariance matrix to use. If `coef.indices` contains a zero, an extra row and column are included at the indicated position, to represent the zero variances and covariances associated with a reference level. If `coef.indices` is `NULL`, then `factorname` should be the name of a factor effect in the model, and is used in order to extract the necessary variance-covariance estimates.

Quasi-variances were first implemented in R with **qvcalc**. This implementation draws heavily from that.

Value

A L by L matrix whose i-j element is the logarithm of the variance of the ith coefficient minus the jth coefficient, for all values of i and j. The diagonal elements are abitrary and are set to zero.

The matrix has an attribute that corresponds to the prior weight matrix; it is accessed by `uninormal` and replaces the usual `weights` argument. of `vglm`. This weight matrix has ones on the off-diagonals and some small positive number on the diagonals.

Warning

Negative quasi-variances may occur (one of them and only one), though they are rare in practice. If so then numerical problems may occur. See `qvcalc()` for more information.

Note

This is an adaptation of `qvcalc()` in **qvcalc**. It should work for all `vglm` models with one linear predictor, i.e., $M = 1$. For $M > 1$ the factor should appear only in one of the linear predictors.

It is important to set `maxit` to be larger than usual for `rcim` since convergence is slow. Upon successful convergence the ith row effect and the ith column effect should be equal. A simple computation involving the fitted and predicted values allows the quasi-variances to be extracted (see example below).

A function to plot *comparison intervals* has not been written here.

Author(s)

T. W. Yee, based heavily on `qvcalc()` in **qvcalc** written by David Firth.

References

Firth, D. (2003). Overcoming the reference category problem in the presentation of statistical models. *Sociological Methodology* **33**, 1–18.

Firth, D. and de Menezes, R. X. (2004). Quasi-variances. *Biometrika* **91**, 65–80.

Yee, T. W. and Hadi, A. F. (2014). Row-column interaction models, with an R implementation. *Computational Statistics*, **29**, 1427–1445.

See Also

`rcim`, `vglm`, `qvar`, `uninormal`, `explink`, `qvcalc()` in **qvcalc**, `ships`.

Examples

```
# Example 1
data("ships", package = "MASS")

Shipmodel <- vglm(incidents ~ type + year + period,
                  poissonff, offset = log(service),
#                 trace = TRUE, model = TRUE,
                  data = ships, subset = (service > 0))
```

```
# Easiest form of input
fit1 <- rcim(Qvar(Shipmodel, "type"), uninormal("explink"), maxit = 99)
qvar(fit1)              # Easy method to get the quasi-variances
qvar(fit1, se = TRUE)   # Easy method to get the quasi-standard errors

(quasiVar <- exp(diag(fitted(fit1))) / 2)                 # Version 1
(quasiVar <- diag(predict(fit1)[, c(TRUE, FALSE)]) / 2)   # Version 2
(quasiSE  <- sqrt(quasiVar))

# Another form of input
fit2 <- rcim(Qvar(Shipmodel, coef.ind = c(0, 2:5), reference.name = "typeA"),
             uninormal("explink"), maxit = 99)
## Not run:  qvplot(fit2, col = "green", lwd = 3, scol = "blue", slwd = 2, las = 1)

# The variance-covariance matrix is another form of input (not recommended)
fit3 <- rcim(Qvar(cbind(0, rbind(0, vcov(Shipmodel)[2:5, 2:5])),
                  labels = c("typeA", "typeB", "typeC", "typeD", "typeE"),
                  estimates = c(typeA = 0, coef(Shipmodel)[2:5])),
             uninormal("explink"), maxit = 99)
(QuasiVar <- exp(diag(fitted(fit3))) / 2)                 # Version 1
(QuasiVar <- diag(predict(fit3)[, c(TRUE, FALSE)]) / 2)   # Version 2
(QuasiSE  <- sqrt(quasiVar))
## Not run:  qvplot(fit3)


# Example 2: a model with M > 1 linear predictors
## Not run:  require("VGAMdata")
xs.nz.f <- subset(xs.nz, sex == "F")
xs.nz.f <- subset(xs.nz.f, !is.na(babies)  & !is.na(age) & !is.na(ethnicity))
xs.nz.f <- subset(xs.nz.f, ethnicity != "Other")

clist <- list("sm.bs(age, df = 4)" = rbind(1, 0),
              "sm.bs(age, df = 3)" = rbind(0, 1),
              "ethnicity"          = diag(2),
              "(Intercept)"        = diag(2))
fit1 <- vglm(babies ~ sm.bs(age, df = 4) + sm.bs(age, df = 3) + ethnicity,
            zipoissonff(zero = NULL), xs.nz.f,
            constraints = clist, trace = TRUE)
Fit1 <- rcim(Qvar(fit1, "ethnicity", which.linpred = 1),
             uninormal("explink", imethod = 1), maxit = 99, trace = TRUE)
Fit2 <- rcim(Qvar(fit1, "ethnicity", which.linpred = 2),
             uninormal("explink", imethod = 1), maxit = 99, trace = TRUE)

## End(Not run)
## Not run:  par(mfrow = c(1, 2))
qvplot(Fit1, scol = "blue", pch = 16, main = expression(eta[1]),
       slwd = 1.5, las = 1, length.arrows = 0.07)
qvplot(Fit2, scol = "blue", pch = 16, main = expression(eta[2]),
       slwd = 1.5, las = 1, length.arrows = 0.07)

## End(Not run)
```

qvar *Quasi-variances Extraction Function*

Description

Takes a rcim fit of the appropriate format and returns either the quasi-variances or quasi-standard errors.

Usage

```
qvar(object, se = FALSE, ...)
```

Arguments

object	A rcim object that has family function uninormal with the explink link. See below for an example.
se	Logical. If FALSE then the quasi-variances are returned, else the square root of them, called quasi-standard errors.
...	Currently unused.

Details

This simple function is ad hoc and simply is equivalent to computing the quasi-variances by `diag(predict(fit1)[, c(TRUE, FALSE)]) / 2`. This function is for convenience only. Serious users of quasi-variances ought to understand why and how this function works.

Value

A vector of quasi-variances or quasi-standard errors.

Author(s)

T. W. Yee.

See Also

rcim, uninormal, explink, Qvar, ships.

Examples

```
data("ships", package = "MASS")
Shipmodel <- vglm(incidents ~ type + year + period,
                  poissonff, offset = log(service),
                  data = ships, subset = (service > 0))

# Easiest form of input
fit1 = rcim(Qvar(Shipmodel, "type"), uninormal("explink"), maxit=99)
qvar(fit1)             # Quasi-variances
```

```
qvar(fit1, se = TRUE)  # Quasi-standard errors

# Manually compute them:
(quasiVar <- exp(diag(fitted(fit1))) / 2)                # Version 1
(quasiVar <- diag(predict(fit1)[, c(TRUE, FALSE)]) / 2)  # Version 2
(quasiSE  <- sqrt(quasiVar))

## Not run:  qvplot(fit1, col = "green", lwd = 3, scol = "blue",
     slwd = 2, las = 1)
## End(Not run)
```

R2latvar *R-squared for Latent Variable Models*

Description

R-squared goodness of fit for latent variable models, such as cumulative link models. Some software such as Stata call the quantity the McKelvey–Zavoina R-squared, which was proposed in their 1975 paper for cumulative probit models.

Usage

```
R2latvar(object)
```

Arguments

object	A `cumulative` or `binomialff` fit using `vglm`. Only a few selected link functions are currently permitted: `logitlink`, `probitlink`, `clogloglink`. For models with more than one linear predictor, a parallelism assumption is needed also, i.e., the constraint matrices must be a 1-column matrix of 1s (except for the intercept). The model is assumed to have an intercept term.

Details

Models such as the proportional odds model have a latent variable interpretation (see, e.g., Section 6.2.6 of Agresti (2018), Section 14.4.1.1 of Yee (2015), Section 5.2.2 of McCullagh and Nelder (1989)). It is possible to summarize the predictive power of the model by computing R^2 on the transformed scale, e.g., on a standard normal distribution for a `probitlink` link. For more details see Section 6.3.7 of Agresti (2018).

Value

The R^2 value. Approximately, that amount is the variability in the latent variable of the model explained by all the explanatory variables. Then taking the positive square-root gives an approximate multiple correlation R.

Author(s)

Thomas W. Yee

References

Agresti, A. (2018). *An Introduction to Categorical Data Analysis, 3rd ed.*, New York: John Wiley & Sons.

McKelvey, R. D. and W. Zavoina (1975). A statistical model for the analysis of ordinal level dependent variables. *The Journal of Mathematical Sociology*, **4**, 103–120.

See Also

`vglm`, `cumulative`, `propodds`, `logitlink`, `probitlink`, `clogloglink`, `summary.lm`.

Examples

```
pneumo <- transform(pneumo, let = log(exposure.time))
(fit <- vglm(cbind(normal, mild, severe) ~ let, propodds, data = pneumo))
R2latvar(fit)
```

Rank *Rank*

Description

Returns the rank of reduced-rank regression-type models in the VGAM package.

Usage

```
Rank(object, ...)
Rank.rrvglm(object, ...)
Rank.qrrvglm(object, ...)
Rank.rrvgam(object, ...)
```

Arguments

`object`	Some **VGAM** object, for example, having class `rrvglm-class`. The class `vglm-class` is not included since this is not based on reduced-rank regression.
`...`	Other possible arguments fed into the function later (used for added flexibility for the future).

Details

Regression models based on reduced-rank regression have a quantity called the *rank*, which is 1 or 2 or 3 etc. The smaller the value the more dimension reduction, so that there are fewer parameters. This function was not called `rank()` to avoid conflict with `rank`.

Value

Returns an integer value, provided the rank of the model makes sense.

Note

This function has not been defined for VGLMs yet. It might refer to the rank of the VL model matrix, but for now this function should not be applied to vglm fits.

Author(s)

T. W. Yee.

See Also

RR-VGLMs are described in rrvglm-class; QRR-VGLMs are described in qrrvglm-class.

Examples

```
pneumo <- transform(pneumo, let = log(exposure.time),
                            x3  = runif(nrow(pneumo)))
(fit1 <- rrvglm(cbind(normal, mild, severe) ~ let + x3,
                acat, data = pneumo))
coef(fit1, matrix = TRUE)
constraints(fit1)
Rank(fit1)
```

Rayleigh *Rayleigh Distribution*

Description

Density, distribution function, quantile function and random generation for the Rayleigh distribution with parameter a.

Usage

```
drayleigh(x, scale = 1, log = FALSE)
prayleigh(q, scale = 1, lower.tail = TRUE, log.p = FALSE)
qrayleigh(p, scale = 1, lower.tail = TRUE, log.p = FALSE)
rrayleigh(n, scale = 1)
```

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
n	number of observations. Fed into runif.
scale	the scale parameter b.
log	Logical. If log = TRUE then the logarithm of the density is returned.
lower.tail, log.p	Same meaning as in pnorm or qnorm.

Details

See `rayleigh`, the **VGAM** family function for estimating the scale parameter b by maximum likelihood estimation, for the formula of the probability density function and range restrictions on the parameter b.

Value

`drayleigh` gives the density, `prayleigh` gives the distribution function, `qrayleigh` gives the quantile function, and `rrayleigh` generates random deviates.

Note

The Rayleigh distribution is related to the Maxwell distribution.

Author(s)

T. W. Yee and Kai Huang

References

Forbes, C., Evans, M., Hastings, N. and Peacock, B. (2011). *Statistical Distributions*, Hoboken, NJ, USA: John Wiley and Sons, Fourth edition.

See Also

`rayleigh`, `maxwell`.

Examples

```
## Not run:  Scale <- 2; x <- seq(-1, 8, by = 0.1)
plot(x, drayleigh(x, scale = Scale), type = "l", ylim = c(0,1),
  las = 1, ylab = "",
  main = "Rayleigh density divided into 10 equal areas; red = CDF")
abline(h = 0, col = "blue", lty = 2)
qq <- qrayleigh(seq(0.1, 0.9, by = 0.1), scale = Scale)
lines(qq, drayleigh(qq, scale = Scale), col = 2, lty = 3, type = "h")
lines(x, prayleigh(x, scale = Scale), col = "red")
## End(Not run)
```

`rayleigh` *Rayleigh Regression Family Function*

Description

Estimating the parameter of the Rayleigh distribution by maximum likelihood estimation. Right-censoring is allowed.

Usage

```
rayleigh(lscale = "loglink", nrfs = 1/3 + 0.01,
         oim.mean = TRUE, zero = NULL, parallel = FALSE,
         type.fitted = c("mean", "percentiles", "Qlink"),
         percentiles = 50)
cens.rayleigh(lscale = "loglink", oim = TRUE)
```

Arguments

`lscale`	Parameter link function applied to the scale parameter b. See `Links` for more choices. A log link is the default because b is positive.
`nrfs`	Numeric, of length one, with value in $[0, 1]$. Weighting factor between Newton-Raphson and Fisher scoring. The value 0 means pure Newton-Raphson, while 1 means pure Fisher scoring. The default value uses a mixture of the two algorithms, and retaining positive-definite working weights.
`oim.mean`	Logical, used only for intercept-only models. `TRUE` means the mean of the OIM elements are used as working weights. If `TRUE` then this argument has top priority for working out the working weights. `FALSE` means use another algorithm.
`oim`	Logical. For censored data only, `TRUE` means the Newton-Raphson algorithm, and `FALSE` means Fisher scoring.
`zero, parallel`	Details at `CommonVGAMffArguments`.
`type.fitted, percentiles`	See `CommonVGAMffArguments` for information. Using `"Qlink"` is for quantile-links in **VGAMextra**.

Details

The Rayleigh distribution, which is used in physics, has a probability density function that can be written

$$f(y) = y \exp(-0.5(y/b)^2)/b^2$$

for $y > 0$ and $b > 0$. The mean of Y is $b\sqrt{\pi/2}$ (returned as the fitted values) and its variance is $b^2(4 - \pi)/2$.

The **VGAM** family function `cens.rayleigh` handles right-censored data (the true value is greater than the observed value). To indicate which type of censoring, input `extra = list(rightcensored = vec2)` where `vec2` is a logical vector the same length as the response. If the component of this list is missing then the logical values are taken to be `FALSE`. The fitted object has this component stored in the `extra` slot.

The **VGAM** family function `rayleigh` handles multiple responses.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, `rrvglm` and `vgam`.

Warning

The theory behind the argument `oim` is not fully complete.

Note

The `poisson.points` family function is more general so that if `ostatistic` = 1 and `dimension` = 2 then it coincides with `rayleigh`. Other related distributions are the Maxwell and Weibull distributions.

Author(s)

T. W. Yee

References

Forbes, C., Evans, M., Hastings, N. and Peacock, B. (2011). *Statistical Distributions*, Hoboken, NJ, USA: John Wiley and Sons, Fourth edition.

See Also

`Rayleigh`, `genrayleigh`, `riceff`, `maxwell`, `weibullR`, `poisson.points`, `simulate.vlm`.

Examples

```
nn <- 1000; Scale <- exp(2)
rdata <- data.frame(ystar = rrayleigh(nn, scale = Scale))
fit <- vglm(ystar ~ 1, rayleigh, data = rdata, trace = TRUE)
head(fitted(fit))
with(rdata, mean(ystar))
coef(fit, matrix = TRUE)
Coef(fit)

# Censored data
rdata <- transform(rdata, U = runif(nn, 5, 15))
rdata <- transform(rdata, y = pmin(U, ystar))
## Not run:  par(mfrow = c(1, 2))
hist(with(rdata, ystar)); hist(with(rdata, y))
## End(Not run)
extra <- with(rdata, list(rightcensored = ystar > U))
fit <- vglm(y ~ 1, cens.rayleigh, data = rdata, trace = TRUE,
            extra = extra, crit = "coef")
table(fit@extra$rightcen)
coef(fit, matrix = TRUE)
head(fitted(fit))
```

Rcim *Mark the Baseline of Row and Column on a Matrix data*

Description

Rearrange the rows and columns of the input so that the first row and first column are baseline. This function is for rank-zero row-column interaction models (RCIMs; i.e., general main effects models).

Usage

```
Rcim(mat, rbaseline = 1, cbaseline = 1)
```

Arguments

`mat` Matrix, of dimension r by c. It is best that it is labelled with row and column names.

`rbaseline, cbaseline` Numeric (row number of the matrix `mat`) or character (matching a row name of `mat`) that the user wants as the row baseline or reference level. Similarly `cbaseline` for the column.

Details

This is a data preprocessing function for `rcim`. For rank-zero row-column interaction models this function establishes the baseline (or reference) levels of the matrix response with respect to the row and columns—these become the new first row and column.

Value

Matrix of the same dimension as the input, with `rbaseline` and `cbaseline` specifying the first rows and columns. The default is no change in `mat`.

Note

This function is similar to `moffset`; see `moffset` for information about the differences. If numeric, the arguments `rbaseline` and `cbaseline` differ from arguments `roffset` and `coffset` in `moffset` by 1 (when elements of the matrix agree).

Author(s)

Alfian F. Hadi and T. W. Yee.

See Also

`moffset`, `rcim`, `plotrcim0`.

Examples

```
(alcoff.e <- moffset(alcoff, roffset = "6", postfix = "*"))
(aa <- Rcim(alcoff,    rbaseline = "11", cbaseline = "Sun"))
(bb <- moffset(alcoff,             "11",             "Sun", postfix = "*"))
aa - bb  # Note the difference!
```

rcqo *Constrained Quadratic Ordination*

Description

Random generation for constrained quadratic ordination (CQO).

Usage

```
rcqo(n, p, S, Rank = 1,
     family = c("poisson", "negbinomial", "binomial-poisson",
                "Binomial-negbinomial", "ordinal-poisson",
                "Ordinal-negbinomial", "gamma2"),
     eq.maximums = FALSE, eq.tolerances = TRUE, es.optimums = FALSE,
     lo.abundance = if (eq.maximums) hi.abundance else 10,
     hi.abundance = 100, sd.latvar = head(1.5/2^(0:3), Rank),
     sd.optimums = ifelse(es.optimums, 1.5/Rank, 1) *
                   ifelse(scale.latvar, sd.latvar, 1),
     sd.tolerances = 0.25, Kvector = 1, Shape = 1,
     sqrt.arg = FALSE, log.arg = FALSE, rhox = 0.5, breaks = 4,
     seed = NULL, optimums1.arg = NULL, Crow1positive = TRUE,
     xmat = NULL, scale.latvar = TRUE)
```

Arguments

`n` Number of sites. It is denoted by n below.

`p` Number of environmental variables, including an intercept term. It is denoted by p below. Must be no less than $1 + R$ in value.

`S` Number of species. It is denoted by S below.

`Rank` The rank or the number of latent variables or true dimension of the data on the reduced space. This must be either 1, 2, 3 or 4. It is denoted by R.

`family` What type of species data is to be returned. The first choice is the default. If binomial then a 0 means absence and 1 means presence. If ordinal then the `breaks` argument is passed into the `breaks` argument of `cut`. Note that either the Poisson or negative binomial distributions are used to generate binomial and ordinal data, and that an upper-case choice is used for the negative binomial distribution (this makes it easier for the user). If `"gamma2"` then this is the 2-parameter gamma distribution.

`eq.maximums` Logical. Does each species have the same maximum? See arguments `lo.abundance` and `hi.abundance`.

`eq.tolerances` Logical. Does each species have the same tolerance? If `TRUE` then the common value is 1 along every latent variable, i.e., all species' tolerance matrices are the order-R identity matrix.

es.optimums
: Logical. Do the species have equally spaced optimums? If TRUE then the quantity $S^{1/R}$ must be an integer with value 2 or more. That is, there has to be an appropriate number of species in total. This is so that a grid of optimum values is possible in R-dimensional latent variable space in order to place the species' optimums. Also see the argument sd.tolerances.

lo.abundance, hi.abundance
: Numeric. These are recycled to a vector of length S. The species have a maximum between lo.abundance and hi.abundance. That is, at their optimal environment, the mean abundance of each species is between the two componentwise values. If eq.maximums is TRUE then lo.abundance and hi.abundance must have the same values. If eq.maximums is FALSE then the logarithm of the maximums are uniformly distributed between log(lo.abundance) and log(hi.abundance).

sd.latvar
: Numeric, of length R (recycled if necessary). Site scores along each latent variable have these standard deviation values. This must be a decreasing sequence of values because the first ordination axis contains the greatest spread of the species' site scores, followed by the second axis, followed by the third axis, etc.

sd.optimums
: Numeric, of length R (recycled if necessary). If es.optimums = FALSE then, for the rth latent variable axis, the optimums of the species are generated from a normal distribution centered about 0. If es.optimums = TRUE then the S optimums are equally spaced about 0 along every latent variable axis. Regardless of the value of es.optimums, the optimums are then scaled to give standard deviation sd.optimums[r].

sd.tolerances
: Logical. If eq.tolerances = FALSE then, for the rth latent variable, the species' tolerances are chosen from a normal distribution with mean 1 and standard deviation sd.tolerances[r]. However, the first species y1 has its tolerance matrix set equal to the order-R identity matrix. All tolerance matrices for all species are diagonal in this function. This argument is ignored if eq.tolerances is TRUE, otherwise it is recycled to length R if necessary.

Kvector
: A vector of positive k values (recycled to length S if necessary) for the negative binomial distribution (see negbinomial for details). Note that a natural default value does not exist, however the default value here is probably a realistic one, and that for large values of μ one has $Var(Y) = \mu^2/k$ approximately.

Shape
: A vector of positive λ values (recycled to length S if necessary) for the 2-parameter gamma distribution (see gamma2 for details). Note that a natural default value does not exist, however the default value here is probably a realistic one, and that $Var(Y) = \mu^2/\lambda$.

sqrt.arg
: Logical. Take the square-root of the negative binomial counts? Assigning sqrt.arg = TRUE when family="negbinomial" means that the resulting species data can be considered very crudely to be approximately Poisson distributed. They will not integers in general but much easier (less numerical problems) to estimate using something like cqo(..., family="poissonff").

log.arg
: Logical. Take the logarithm of the gamma random variates? Assigning log.arg = TRUE when family="gamma2" means that the resulting species data can be considered very crudely to be approximately Gaussian distributed about its (quadratic) mean.

rhox
: Numeric, less than 1 in absolute value. The correlation between the environmental variables. The correlation matrix is a matrix of 1's along the diagonal and

rhox in the off-diagonals. Note that each environmental variable is normally distributed with mean 0. The standard deviation of each environmental variable is chosen so that the site scores have the determined standard deviation, as given by argument sd.latvar.

breaks
: If family is assigned an ordinal value then this argument is used to define the cutpoints. It is fed into the breaks argument of cut.

seed
: If given, it is passed into set.seed. This argument can be used to obtain reproducible results. If set, the value is saved as the "seed" attribute of the returned value. The default will not change the random generator state, and return .Random.seed as "seed" attribute.

optimums1.arg
: If assigned and Rank = 1 then these are the explicity optimums. Recycled to length S.

Crow1positive
: See qrrvglm.control for details.

xmat
: The n by $p-1$ environmental matrix can be inputted.

scale.latvar
: Logical. If FALSE the argument sd.latvar is ignored and no scaling of the latent variable values is performed.

Details

This function generates data coming from a constrained quadratic ordination (CQO) model. In particular, data coming from a *species packing model* can be generated with this function. The species packing model states that species have equal tolerances, equal maximums, and optimums which are uniformly distributed over the latent variable space. This can be achieved by assigning the arguments es.optimums = TRUE, eq.maximums = TRUE, eq.tolerances = TRUE.

At present, the Poisson and negative binomial abundances are generated first using lo.abundance and hi.abundance, and if family is binomial or ordinal then it is converted into these forms.

In CQO theory the n by p matrix X is partitioned into two parts X_1 and X_2. The matrix X_2 contains the 'real' environmental variables whereas the variables in X_1 are just for adjustment purposes; they contain the intercept terms and other variables that one wants to adjust for when (primarily) looking at the variables in X_2. This function has X_1 only being a matrix of ones, i.e., containing an intercept only.

Value

A n by $p-1+S$ data frame with components and attributes. In the following the attributes are labelled with double quotes.

x2, x3, x4, ..., xp
: The environmental variables. This makes up the n by $p-1$ X_2 matrix. Note that x1 is not present; it is effectively a vector of ones since it corresponds to an intercept term when cqo is applied to the data.

y1, y2, x3, ..., yS
: The species data. This makes up the n by S matrix Y. This will be of the form described by the argument family.

"concoefficients"
: The $p-1$ by R matrix of constrained coefficients (or canonical coefficients). These are also known as weights or loadings.

"formula"	The formula involving the species and environmental variable names. This can be used directly in the formula argument of cqo.
"log.maximums"	The S-vector of species' maximums, on a log scale. These are uniformly distributed between log(lo.abundance) and log(hi.abundance).
"latvar"	The n by R matrix of site scores. Each successive column (latent variable) has sample standard deviation equal to successive values of sd.latvar.
"eta"	The linear/additive predictor value.
"optimums"	The S by R matrix of species' optimums.
"tolerances"	The S by R matrix of species' tolerances. These are the square root of the diagonal elements of the tolerance matrices (recall that all tolerance matrices are restricted to being diagonal in this function).

Other attributes are "break", "family", "Rank", "lo.abundance", "hi.abundance", "eq.tolerances", "eq.maximums", "seed" as used.

Note

This function is under development and is not finished yet. There may be a few bugs.

Yet to do: add an argument that allows absences to be equal to the first level if ordinal data is requested.

Author(s)

T. W. Yee

References

Yee, T. W. (2004). A new technique for maximum-likelihood canonical Gaussian ordination. *Ecological Monographs*, **74**, 685–701.

Yee, T. W. (2006). Constrained additive ordination. *Ecology*, **87**, 203–213.

ter Braak, C. J. F. and Prentice, I. C. (1988). A theory of gradient analysis. *Advances in Ecological Research*, **18**, 271–317.

See Also

cqo, qrrvglm.control, cut, binomialff, poissonff, negbinomial, gamma2.

Examples

```
## Not run:
# Example 1: Species packing model:
n <- 100; p <- 5; S <- 5
mydata <- rcqo(n, p, S, es.opt = TRUE, eq.max = TRUE)
names(mydata)
(myform <- attr(mydata, "formula"))
fit <- cqo(myform, poissonff, mydata, Bestof = 3)  # eq.tol = TRUE
matplot(attr(mydata, "latvar"), mydata[,-(1:(p-1))], col = 1:S)
persp(fit, col = 1:S, add = TRUE)
```

```
lvplot(fit, lcol = 1:S, y = TRUE, pcol = 1:S)  # Same plot as above

# Compare the fitted model with the 'truth'
concoef(fit)  # The fitted model
attr(mydata, "concoefficients")  # The 'truth'

c(apply(attr(mydata, "latvar"), 2, sd),
  apply(latvar(fit), 2, sd))  # Both values should be approx equal


# Example 2: negative binomial data fitted using a Poisson model:
n <- 200; p <- 5; S <- 5
mydata <- rcqo(n, p, S, fam = "negbin", sqrt = TRUE)
myform <- attr(mydata, "formula")
fit <- cqo(myform, fam = poissonff, dat = mydata)  # I.tol = TRUE,
lvplot(fit, lcol = 1:S, y = TRUE, pcol = 1:S)
# Compare the fitted model with the 'truth'
concoef(fit)  # The fitted model
attr(mydata, "concoefficients")  # The 'truth'

## End(Not run)
```

---

rdiric *The Dirichlet distribution*

Description

Generates Dirichlet random variates.

Usage

```
rdiric(n, shape, dimension = NULL, is.matrix.shape = FALSE)
```

Arguments

n	number of observations. Note it has two meanings, see `is.matrix.shape` below.
shape	the shape parameters. These must be positive. If `dimension` is specifed, values are recycled if necessary to length `dimension`.
dimension	the dimension of the distribution. If `dimension` is not numeric then it is taken to be `length(shape)` (or `ncol(shape)` if `is.matrix.shape == TRUE`).
is.matrix.shape	Logical. If `TRUE` then `shape` must be a matrix, and then `n` is no longer the number of rows of the answer but the answer has `n * nrow(shape)` rows. If `FALSE` (the default) then `shape` is a vector and each of the `n` rows of the answer have `shape` as its shape parameters.

Details

This function is based on a relationship between the gamma and Dirichlet distribution. Random gamma variates are generated, and then Dirichlet random variates are formed from these.

Value

A `n` by `dimension` matrix of Dirichlet random variates. Each element is positive, and each row will sum to unity. If `shape` has names then these will become the column names of the answer.

Author(s)

Thomas W. Yee

References

Lange, K. (2002). *Mathematical and Statistical Methods for Genetic Analysis*, 2nd ed. New York: Springer-Verlag.

See Also

`dirichlet` is a **VGAM** family function for fitting a Dirichlet distribution to data.

Examples

```
ddata <- data.frame(rdiric(n = 1000, shape = c(y1 = 3, y2 = 1, y3 = 4)))
fit <- vglm(cbind(y1, y2, y3) ~ 1, dirichlet, data = ddata, trace = TRUE)
Coef(fit)
coef(fit, matrix = TRUE)
```

`rec.exp1` *Upper Record Values from a 1-parameter Exponential Distribution*

Description

Maximum likelihood estimation of the rate parameter of a 1-parameter exponential distribution when the observations are upper record values.

Usage

```
rec.exp1(lrate = "loglink", irate = NULL, imethod = 1)
```

Arguments

`lrate`	Link function applied to the rate parameter. See `Links` for more choices.
`irate`	Numeric. Optional initial values for the rate. The default value `NULL` means they are computed internally, with the help of `imethod`.
`imethod`	Integer, either 1 or 2 or 3. Initial method, three algorithms are implemented. Choose the another value if convergence fails, or use `irate`.

Details

The response must be a vector or one-column matrix with strictly increasing values.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Note

By default, this family function has the intercept-only MLE as the initial value, therefore convergence may only take one iteration. Fisher scoring is used.

Author(s)

T. W. Yee

References

Arnold, B. C. and Balakrishnan, N. and Nagaraja, H. N. (1998). *Records*, New York: John Wiley & Sons.

See Also

exponential.

Examples

```
rawy <- rexp(n <- 10000, rate = exp(1))
y <- unique(cummax(rawy))  # Keep only the records

length(y) / y[length(y)]   # MLE of rate

fit <- vglm(y ~ 1, rec.exp1, trace = TRUE)
coef(fit, matrix = TRUE)
Coef(fit)
```

---

rec.normal   *Upper Record Values from a Univariate Normal Distribution*

Description

Maximum likelihood estimation of the two parameters of a univariate normal distribution when the observations are upper record values.

Usage

```
rec.normal(lmean = "identitylink", lsd = "loglink",
           imean = NULL, isd = NULL, imethod = 1, zero = NULL)
```

Arguments

- `lmean, lsd` — Link functions applied to the mean and sd parameters. See `Links` for more choices.
- `imean, isd` — Numeric. Optional initial values for the mean and sd. The default value `NULL` means they are computed internally, with the help of `imethod`.
- `imethod` — Integer, either 1 or 2 or 3. Initial method, three algorithms are implemented. Choose the another value if convergence fails, or use `imean` and/or `isd`.
- `zero` — Can be an integer vector, containing the value 1 or 2. If so, the mean or standard deviation respectively are modelled as an intercept only. Usually, setting `zero = 2` will be used, if used at all. The default value `NULL` means both linear/additive predictors are modelled as functions of the explanatory variables. See `CommonVGAMffArguments` for more information.

Details

The response must be a vector or one-column matrix with strictly increasing values.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Note

This family function tries to solve a difficult problem, and the larger the data set the better. Convergence failure can commonly occur, and convergence may be very slow, so set `maxit = 200, trace = TRUE`, say. Inputting good initial values are advised.

This family function uses the BFGS quasi-Newton update formula for the working weight matrices. Consequently the estimated variance-covariance matrix may be inaccurate or simply wrong! The standard errors must be therefore treated with caution; these are computed in functions such as `vcov()` and `summary()`.

Author(s)

T. W. Yee

References

Arnold, B. C. and Balakrishnan, N. and Nagaraja, H. N. (1998). *Records*, New York: John Wiley & Sons.

See Also

`uninormal`, `double.cens.normal`.

Examples

```
nn <- 10000; mymean <- 100
# First value is reference value or trivial record
Rdata <- data.frame(rawy = c(mymean, rnorm(nn, mymean, exp(3))))
# Keep only observations that are records:
rdata <- data.frame(y = unique(cummax(with(Rdata, rawy))))

fit <- vglm(y ~ 1, rec.normal, rdata, trace = TRUE, maxit = 200)
coef(fit, matrix = TRUE)
Coef(fit)
summary(fit)
```

`reciprocallink` *Reciprocal Link Function*

Description

Computes the reciprocal transformation, including its inverse and the first two derivatives.

Usage

```
   reciprocallink(theta, bvalue = NULL, inverse = FALSE, deriv = 0,
                  short = TRUE, tag = FALSE)
negreciprocallink(theta, bvalue = NULL, inverse = FALSE, deriv = 0,
                  short = TRUE, tag = FALSE)
```

Arguments

`theta` Numeric or character. See below for further details.

`bvalue` See `Links`.

`inverse, deriv, short, tag`
Details at `Links`.

Details

The `reciprocallink` link function is a special case of the power link function. Numerical values of `theta` close to 0 result in `Inf`, `-Inf`, `NA` or `NaN`.

The `negreciprocallink` link function computes the negative reciprocal, i.e., $-1/\theta$.

Value

For `reciprocallink`: for `deriv = 0`, the reciprocal of `theta`, i.e., `1/theta` when `inverse = FALSE`, and if `inverse = TRUE` then `1/theta`. For `deriv = 1`, then the function returns *d* `theta` / *d* `eta` as a function of `theta` if `inverse = FALSE`, else if `inverse = TRUE` then it returns the reciprocal.

Note

Numerical instability may occur when `theta` is close to 0.

Author(s)

Thomas W. Yee

References

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*, 2nd ed. London: Chapman & Hall.

See Also

`identitylink`, `powerlink`.

Examples

```
   reciprocallink(1:5)
   reciprocallink(1:5, inverse = TRUE, deriv = 2)
negreciprocallink(1:5)
negreciprocallink(1:5, inverse = TRUE, deriv = 2)

x <- (-3):3
reciprocallink(x)  # Has Inf
reciprocallink(x, bvalue = .Machine$double.eps)  # Has no Inf
```

---

residualsvglm *Residuals for a VGLM fit*

Description

Residuals for a vector generalized linear model (VGLM) object.

Usage

```
residualsvglm(object, type = c("working", "pearson", "response",
   "deviance", "ldot", "stdres", "rquantile"), matrix.arg = TRUE)
```

Arguments

`object`	Object of class `"vglm"`, i.e., a `vglm` fit.
`type`	The value of this argument can be abbreviated. The type of residuals to be returned. The default is the first one: working residuals corresponding to the IRLS algorithm. These are defined for all models. They are sometimes added to VGAM plots of estimated component functions (see `plotvgam`). Pearson residuals for GLMs, when squared and summed over the data set, total to the Pearson chi-squared statistic. For VGLMs, Pearson residuals involve the working weight matrices and the score vectors. Under certain limiting conditions, Pearson residuals have 0 means and identity matrix as the variance-covariance matrix.

Response residuals are simply the difference between the observed values and the fitted values. Both have to be of the same dimension, hence not all families have response residuals defined.

Deviance residuals are only defined for models with a deviance function. They tend to GLMs mainly. This function returns a `NULL` for those models whose deviance is undefined.

Randomized quantile residuals (RQRs) (Dunn and Smyth, 1996) are based on the p-type function being fed into `qnorm`. For example, for the default `exponential` it is `qnorm(pexp(y, rate = 1 / fitted(object)))`. So one should expect these residuals to have a standard normal distribution if the model and data agree well. If the distribution is discrete then *randomized* values are returned; see `runif` and `set.seed`. For example, for the default `poissonff` it is `qnorm(runif(length(y), ppois(y - 1, mu), ppois(y, mu)))` where `mu` is the fitted mean. The following excerpts comes from their writings. They highly recommend quantile residuals for discrete distributions since plots using deviance and Pearson residuals may contain distracting patterns. Four replications of the quantile residuals are recommended with discrete distributions because they have a random component. Any features not preserved across all four sets of residuals are considered artifacts of the randomization. This type of residual is continuous even for discrete distributions; for both discrete and continuous distributions, the quantile residuals have an exact standard normal distribution.

The choice `"ldot"` should not be used currently.

Standardized residuals are currently only defined for 2 types of models: (i) GLMs (`poissonff`, `binomialff`); (ii) those fitted to a two-way table of counts, e.g., `cumulative`, `acat`, `multinomial`, `sratio`, `cratio`. For (ii), they are defined in Section 2.4.5 of Agresti (2018) and are also the output from the `"stdres"` component of `chisq.test`. For the test of independence they are a useful type of residual. Their formula is `(observed - expected) / sqrt(V)`, where `V` is the residual cell variance (also see Agresti, 2007, section 2.4.5). When an independence null hypothesis is true, each standardized residual (corresponding to a cell in the table) has a a large-sample standard normal distribution. Currently this function merely extracts the table of counts from `object` and then computes the standardized residuals like `chisq.test`.

`matrix.arg` Logical, which applies when if the pre-processed answer is a vector or a 1-column matrix. If `TRUE` then the value returned will be a matrix, else a vector.

Details

This function returns various kinds of residuals, sometimes depending on the specific type of model having been fitted. Section 3.7 of Yee (2015) gives some details on several types of residuals defined for the VGLM class.

Standardized residuals for GLMs are described in Section 4.5.6 of Agresti (2013) as the ratio of the raw (response) residuals divided by their standard error. They involve the generalized hat matrix evaluated at the final IRLS iteration. When applied to the LM, standardized residuals for GLMs simplify to `rstandard`. For GLMs they are basically the Pearson residual divided by the square root of 1 minus the leverage.

Value

If that residual type is undefined or inappropriate or not yet implemented, then `NULL` is returned, otherwise a matrix or vector of residuals is returned.

Warning

This function may change in the future, especially those whose definitions may change.

References

Agresti, A. (2007). *An Introduction to Categorical Data Analysis, 2nd ed.*, New York: John Wiley & Sons. Page 38.

Agresti, A. (2013). *Categorical Data Analysis, 3rd ed.*, New York: John Wiley & Sons.

Agresti, A. (2018). *An Introduction to Categorical Data Analysis, 3rd ed.*, New York: John Wiley & Sons.

Dunn, P. K. and Smyth, G. K. (1996). Randomized quantile residuals. *Journal of Computational and Graphical Statistics*, **5**, 236–244.

See Also

`resid`, `vglm`, `chisq.test`, `hatvalues`.

Examples

```
pneumo <- transform(pneumo, let = log(exposure.time))
fit <- vglm(cbind(normal, mild, severe) ~ let, propodds, pneumo)
resid(fit)  # Same as having type = "working" (the default)
resid(fit, type = "response")
resid(fit, type = "pearson")
resid(fit, type = "stdres")  # Test for independence
```

`rhobitlink` *Rhobit Link Function*

Description

Computes the rhobit link transformation, including its inverse and the first two derivatives.

Usage

```
rhobitlink(theta, bminvalue = NULL, bmaxvalue = NULL,
           inverse = FALSE, deriv = 0, short = TRUE, tag = FALSE)
```

Arguments

`theta` Numeric or character. See below for further details.

`bminvalue, bmaxvalue`
Optional boundary values, e.g., values of `theta` which are less than or equal to -1 can be replaced by `bminvalue` before computing the link function value. And values of `theta` which are greater than or equal to 1 can be replaced by `bmaxvalue` before computing the link function value. See `Links`.

`inverse, deriv, short, tag`
Details at `Links`.

Details

The `rhobitlink` link function is commonly used for parameters that lie between -1 and 1. Numerical values of `theta` close to -1 or 1 or out of range result in `Inf`, `-Inf`, `NA` or `NaN`.

Value

For `deriv = 0`, the rhobit of `theta`, i.e., `log((1 + theta)/(1 - theta))` when `inverse = FALSE`, and if `inverse = TRUE` then `(exp(theta) - 1)/(exp(theta) + 1)`.

For `deriv = 1`, then the function returns *d* `eta` / *d* `theta` as a function of `theta` if `inverse = FALSE`, else if `inverse = TRUE` then it returns the reciprocal.

Note

Numerical instability may occur when `theta` is close to -1 or 1. One way of overcoming this is to use `bminvalue`, etc.

The correlation parameter of a standard bivariate normal distribution lies between -1 and 1, therefore this function can be used for modelling this parameter as a function of explanatory variables.

The link function `rhobitlink` is very similar to `fisherzlink`, e.g., just twice the value of `fisherzlink`.

Author(s)

Thomas W. Yee

See Also

`Links`, `binom2.rho`, `fisherz`.

Examples

```
theta <- seq(-0.99, 0.99, by = 0.01)
y <- rhobitlink(theta)
## Not run:
plot(theta, y, type = "l", ylab = "", main = "rhobitlink(theta)")
abline(v = 0, h = 0, lty = 2)

## End(Not run)

x <- c(seq(-1.02, -0.98, by = 0.01), seq(0.97, 1.02, by = 0.01))
```

```
rhobitlink(x)  # Has NAs
rhobitlink(x, bminvalue = -1 + .Machine$double.eps,
              bmaxvalue =  1 - .Machine$double.eps)  # Has no NAs
```

Rice *The Rice Distribution*

Description

Density, distribution function, quantile function and random generation for the Rician distribution.

Usage

```
drice(x, sigma, vee, log = FALSE)
price(q, sigma, vee, lower.tail = TRUE, log.p = FALSE, ...)
qrice(p, sigma, vee, lower.tail = TRUE, log.p = FALSE, ...)
rrice(n, sigma, vee)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. Same as in `runif`.
`vee, sigma`	See `riceff`.
`...`	Other arguments such as `lower.tail`.
`lower.tail, log.p`	Same meaning as in `pnorm` or `qnorm`.
`log`	Logical. If `log` = `TRUE` then the logarithm of the density is returned.

Details

See `riceff`, the **VGAM** family function for estimating the two parameters, for the formula of the probability density function and other details.

Formulas for `price()` and `qrice()` are based on the Marcum-Q function.

Value

`drice` gives the density, `price` gives the distribution function, `qrice` gives the quantile function, and `rrice` generates random deviates.

Author(s)

T. W. Yee and Kai Huang

See Also

`riceff`.

Examples

```
## Not run:  x <- seq(0.01, 7, len = 201)
plot(x, drice(x, vee = 0, sigma = 1), type = "n", las = 1,
     ylab = "",
     main = "Density of Rice distribution for various v values")
sigma <- 1; vee <- c(0, 0.5, 1, 2, 4)
for (ii in 1:length(vee))
  lines(x, drice(x, vee = vee[ii], sigma), col = ii)
legend(x = 5, y = 0.6, legend = as.character(vee),
       col = 1:length(vee), lty = 1)

x <- seq(0, 4, by = 0.01); vee <- 1; sigma <- 1
probs <- seq(0.05, 0.95, by = 0.05)
plot(x, drice(x, vee = vee, sigma = sigma), type = "l",
     main = "Blue is density, orange is CDF", col = "blue",
     ylim = c(0, 1), sub = "Red are 5, 10, ..., 95 percentiles",
     las = 1, ylab = "", cex.main = 0.9)
abline(h = 0:1, col = "black", lty = 2)
Q <- qrice(probs, sigma, vee = vee)
lines(Q, drice(qrice(probs, sigma, vee = vee),
               sigma, vee = vee), col = "red", lty = 3, type = "h")
lines(x, price(x, sigma, vee = vee), type = "l", col = "orange")
lines(Q, drice(Q, sigma, vee = vee), col = "red", lty = 3, type = "h")
lines(Q, price(Q, sigma, vee = vee), col = "red", lty = 3, type = "h")
abline(h = probs, col = "red", lty = 3)
max(abs(price(Q, sigma, vee = vee) - probs))  # Should be 0

## End(Not run)
```

`riceff` *Rice Distribution Family Function*

Description

Estimates the two parameters of a Rice distribution by maximum likelihood estimation.

Usage

```
riceff(lsigma = "loglink", lvee = "loglink", isigma = NULL,
       ivee = NULL, nsimEIM = 100, zero = NULL, nowarning = FALSE)
```

Arguments

`nowarning`	Logical. Suppress a warning? Ignored for **VGAM** 0.9-7 and higher.
`lvee, lsigma`	Link functions for the v and σ parameters. See `Links` for more choices and for general information.
`ivee, isigma`	Optional initial values for the parameters. If convergence failure occurs (this **VGAM** family function seems to require good initial values) try using these arguments. See `CommonVGAMffArguments` for more information.
`nsimEIM, zero`	See `CommonVGAMffArguments` for information.

Details

The Rician distribution has density function

$$f(y; v, \sigma) = \frac{y}{\sigma^2} \exp(-(y^2 + v^2)/(2\sigma^2)) \, I_0(yv/\sigma^2)$$

where $y > 0$, $v > 0$, $\sigma > 0$ and I_0 is the modified Bessel function of the first kind with order zero. When $v = 0$ the Rice distribution reduces to a Rayleigh distribution. The mean is $\sigma\sqrt{\pi/2}\exp(z/2)((1-z)I_0(-z/2) - zI_1(-z/2))$ (returned as the fitted values) where $z = -v^2/(2\sigma^2)$. Simulated Fisher scoring is implemented.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Note

Convergence problems may occur for data where $v = 0$; if so, use `rayleigh` or possibly use an `identity` link.

When v is large (greater than 3, say) then the mean is approximately v and the standard deviation is approximately σ.

Author(s)

T. W. Yee

References

Rice, S. O. (1945). Mathematical Analysis of Random Noise. *Bell System Technical Journal*, **24**, 46–156.

See Also

`drice`, `rayleigh`, `besselI`, `simulate.vlm`.

Examples

```
## Not run:  sigma <- exp(1); vee <- exp(2)
rdata <- data.frame(y = rrice(n <- 1000, sigma, vee = vee))
fit <- vglm(y ~ 1, riceff, data = rdata, trace = TRUE, crit = "c")
c(with(rdata, mean(y)), fitted(fit)[1])
coef(fit, matrix = TRUE)
Coef(fit)
summary(fit)

## End(Not run)
```

rigff — Reciprocal Inverse Gaussian distribution

Description

Estimation of the parameters of a reciprocal inverse Gaussian distribution.

Usage

```
rigff(lmu = "identitylink", llambda = "loglink", imu = NULL,
      ilambda = 1)
```

Arguments

lmu, llambda	Link functions for mu and lambda. See Links for more choices.
imu, ilambda	Initial values for mu and lambda. A NULL means a value is computed internally.

Details

See Jorgensen (1997) for details.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Note

This distribution is potentially useful for dispersion modelling.

Author(s)

T. W. Yee

References

Jorgensen, B. (1997). *The Theory of Dispersion Models*. London: Chapman & Hall

See Also

simplex.

Examples

```
rdata <- data.frame(y = rchisq(100, df = 14))  # Not 'proper' data!!
fit <- vglm(y ~ 1, rigff, rdata, trace = TRUE)
fit <- vglm(y ~ 1, rigff, rdata, trace = TRUE, crit = "c")
summary(fit)
```

rlplot.gevff *Return Level Plot for GEV Fits*

Description

A return level plot is constructed for a GEV-type model.

Usage

```
rlplot.gevff(object, show.plot = TRUE,
    probability = c((1:9)/100, (1:9)/10, 0.95, 0.99, 0.995, 0.999),
    add.arg = FALSE, xlab = if(log.arg) "Return Period (log-scale)" else
    "Return Period", ylab = "Return Level",
    main = "Return Level Plot",
    pch = par()$pch, pcol.arg = par()$col, pcex = par()$cex,
    llty.arg = par()$lty, lcol.arg = par()$col, llwd.arg = par()$lwd,
    slty.arg = par()$lty, scol.arg = par()$col, slwd.arg = par()$lwd,
    ylim = NULL, log.arg = TRUE, CI = TRUE, epsilon = 1e-05, ...)
```

Arguments

`object`	A **VGAM** extremes model of the GEV-type, produced by `vglm` with a family function either `"gev"` or `"gevff"`.
`show.plot`	Logical. Plot it? If `FALSE` no plot will be done.
`probability`	Numeric vector of probabilities used.
`add.arg`	Logical. Add the plot to an existing plot?
`xlab`	Caption for the x-axis. See `par`.
`ylab`	Caption for the y-axis. See `par`.
`main`	Title of the plot. See `title`.
`pch`	Plotting character. See `par`.
`pcol.arg`	Color of the points. See the `col` argument of `par`.
`pcex`	Character expansion of the points. See the `cex` argument of `par`.
`llty.arg`	Line type. Line type. See the `lty` argument of `par`.
`lcol.arg`	Color of the lines. See the `col` argument of `par`.
`llwd.arg`	Line width. See the `lwd` argument of `par`.
`slty.arg, scol.arg, slwd.arg`	Correponding arguments for the lines used for the confidence intervals. Used only if `CI=TRUE`.
`ylim`	Limits for the y-axis. Numeric of length 2.
`log.arg`	Logical. If `TRUE` then `log=""` otherwise `log="x"`. This changes the labelling of the x-axis only.
`CI`	Logical. Add in a 95 percent confidence interval?

`epsilon` Numeric, close to zero. Used for the finite-difference approximation to the first derivatives with respect to each parameter. If too small, numerical problems will occur.

`...` Arguments passed into the `plot` function when setting up the entire plot. Useful arguments here include `sub` and `las`.

Details

A return level plot plots z_p versus $\log(y_p)$. It is linear if the shape parameter $\xi = 0$. If $\xi < 0$ then the plot is convex with asymptotic limit as p approaches zero at $\mu - \sigma/\xi$. And if $\xi > 0$ then the plot is concave and has no finite bound. Here, $G(z_p) = 1 - p$ where $0 < p < 1$ (p corresponds to the argument `probability`) and G is the cumulative distribution function of the GEV distribution. The quantity z_p is known as the *return level* associated with the *return period* $1/p$. For many applications, this means z_p is exceeded by the annual maximum in any particular year with probability p.

The points in the plot are the actual data.

Value

In the `post` slot of the object is a list called `rlplot` with list components

`yp` `-log(probability)`, which is used on the x-axis.

`zp` values which are used for the y-axis

`lower, upper` lower and upper confidence limits for the 95 percent confidence intervals evaluated at the values of `probability` (if `CI=TRUE`).

Note

The confidence intervals are approximate, being based on finite-difference approximations to derivatives.

Author(s)

T. W. Yee

References

Coles, S. (2001). *An Introduction to Statistical Modeling of Extreme Values*. London: Springer-Verlag.

See Also

`gevff`.

Examples

```
gdata <- data.frame(y = rgev(n <- 100, scale = 2, shape = -0.1))
fit <- vglm(y ~ 1, gevff, data = gdata, trace = TRUE)

# Identity link for all parameters:
fit2 <- vglm(y ~ 1, gevff(lshape = identitylink, lscale = identitylink,
                          iscale = 10), data = gdata, trace = TRUE)
coef(fit2, matrix = TRUE)
## Not run:
par(mfrow = c(1, 2))
rlplot(fit) -> i1
rlplot(fit2, pcol = "darkorange", lcol = "blue", log.arg = FALSE,
       scol = "darkgreen", slty = "dashed", las = 1) -> i2
range(i2@post$rlplot$upper - i1@post$rlplot$upper)  # Should be near 0
range(i2@post$rlplot$lower - i1@post$rlplot$lower)  # Should be near 0

## End(Not run)
```

---

rootogram4 *Rootograms (S4 generic) for Assessing Goodness of Fit of Probability Models*

Description

A graphical technique for comparing the observed and fitted counts from a probability model, on a square root scale.

Usage

```
rootogram4(object, ...)
rootogram4vglm(object, newdata = NULL, breaks = NULL, max = NULL,
               xlab = NULL, main = NULL, width = NULL, ...)
```

Arguments

`object`	an object of class `"vglm"`. zz This includes `"vgam"` because `"vlm"` handles both VGLM and VGAM objects.
`newdata`	Data upon which to base the calculations. The default is the one used to fit the model.
`breaks`	numeric. Breaks for the histogram intervals.
`max`	maximum count displayed. If an error message occurs regarding running out of memory then use this argument; it might occur with a very long tailed distribution such as `gaitdzeta`.
`xlab, main`	graphical parameters.
`width`	numeric. Widths of the histogram bars.
`...`	any additional arguments to `rootogram.default` and `plot.rootogram` in **countreg**. Probably the most useful of these are `style = c("hanging", "standing", "suspended")` and `scale = c("sqrt", "raw")`.

Details

Rootograms are a useful graphical technique for comparing the observed counts with the expected counts given a probability model.

This S4 implementation is based very heavily on `rootogram` coming from **countreg**. This package is primarily written by A. Zeileis and C. Kleiber. That package is currently on R-Forge but not CRAN, and it is based on S3. Since **VGAM** is written using S4, it was necessary to define an S4 generic function called `rootogram4()` which dispatches appropriately for S4 objects.

Currently, only a selected number of **VGAM** family functions are implemented. Over time, hopefully more and more will be completed.

Value

See `rootogram` in **countreg**; an object of class `"rootogram0"` inheriting from `"data.frame"` with about 8 variables.

Warning

This function is rudimentary and based totally on the implementation in **countreg**.

Note

The function names used coming from **countreg** have been renamed slightly to avoid conflict.

Author(s)

Package **countreg** is primarily written by A. Zeileis and C. Kleiber. Function `rootogram4()` is based very heavily on **countreg**. T. W. Yee wrote code to unpack variables from many various models and feed them into the appropriate `d`-type function.

References

Friendly, M. and Meyer, D. (2016). *Discrete Data Analysis with R: Visualization and Modeling Techniques for Categorical and Count Data*, Boca Raton, FL, USA: Chapman & Hall/CRC Press.

Kleiber, C. and Zeileis, A. (2016) "Visualizing Count Data Regressions Using Rootograms." *The American Statistician*, **70**(3), 296–303. doi:10.1080/00031305.2016.1173590.

Tukey, J. W. (1977) *Exploratory Data Analysis*, Reading, MA, USA: Addison-Wesley.

See Also

`vglm`, `vgam`, `glm`, `zipoisson`, `zapoisson`, `rootogram` in **countreg**.

Examples

```
## Not run:
data("hspider", package = "VGAM")  # Count responses
hs.p   <- vglm(Pardlugu ~ CoveHerb,   poissonff, data = hspider)
hs.nb  <- vglm(Pardlugu ~ CoveHerb, negbinomial, data = hspider)
hs.zip <- vglm(Pardlugu ~ CoveHerb,   zipoisson, data = hspider)
hs.zap <- vglm(Pardlugu ~ CoveHerb,   zapoisson, data = hspider)
```

```
opar <- par(mfrow = c(2, 2))  # Plot the rootograms
rootogram4(hs.p,   max = 15, main = "poissonff")
rootogram4(hs.nb,  max = 15, main = "negbinomial")
rootogram4(hs.zip, max = 15, main = "zipoisson")
rootogram4(hs.zap, max = 15, main = "zapoisson")
par(opar)

## End(Not run)
```

---

round2 *Rounding of Numbers to Base 2*

Description

'round2' works like 'round' but the rounding has base 2 under consideration so that bits (binary digits) beyond a certain theshold are zeroed.

Usage

```
round2(x, digits10 = 0)
```

Arguments

`x`	Same as `round`.
`digits10`	Same as `digits` in `round`. The `"10"` is to emphasize the usual base 10 used by humans.

Details

`round2()` is intended to allow reliable and safe for `==` comparisons provided both sides have the function applied to the same value of `digits10`. Internally a numeric has its binary representation (bits) past a certain point set to all 0s, while retaining a certain degree of accuracy. Algorithmically, `x` is multiplied by `2^exponent` and then rounded, and then divided by `2^exponent`. The value of `exponent` is approximately `3 * digits10` when `digits10` is positive. If `digits10` is negative then what is returned is `round(x, digits10)`. The value of `exponent` guarantees that `x` has been rounded to at least `digits10` decimal places (often around `digits10 + 1` for safety).

Value

Something similar to `round`.

Author(s)

T. W. Yee.

See Also

`round`, `tobit`.

Examples

```
set.seed(1); x <- sort(rcauchy(10))
x3 <- round2(x, 3)
x3 == round2(x, 3)  # Supposed to be reliable (all TRUE)
rbind(x, x3)  # Comparison
(x3[1]  * 2^(0:9)) / 2^(0:9)
print((x3[1]  * 2^(0:11)), digits = 14)

# Round to approx 1 d.p.
x1 <- round2(x, 1)
x1 == round2(x, 1)  # Supposed to be reliable (all TRUE)
rbind(x, x1)
x1[8] == 0.75  # 3/4
print((x1[1]  * 2^(0:11)), digits = 9)
seq(31) / 32
```

---

rrar *Nested Reduced-rank Autoregressive Models for Multiple Time Series*

Description

Estimates the parameters of a nested reduced-rank autoregressive model for multiple time series.

Usage

```
rrar(Ranks = 1, coefstart = NULL)
```

Arguments

`Ranks`	Vector of integers: the ranks of the model. Each value must be at least one and no more than `M`, where `M` is the number of response variables in the time series. The length of `Ranks` is the *lag*, which is often denoted by the symbol L in the literature.
`coefstart`	Optional numerical vector of initial values for the coefficients. By default, the family function chooses these automatically.

Details

Full details are given in Ahn and Reinsel (1988). Convergence may be very slow, so setting `maxits = 50`, say, may help. If convergence is not obtained, you might like to try inputting different initial values.

Setting `trace = TRUE` in `vglm` is useful for monitoring the progress at each iteration.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Note

This family function should be used within vglm and not with rrvglm because it does not fit into the RR-VGLM framework exactly. Instead, the reduced-rank model is formulated as a VGLM!

A methods function Coef.rrar, say, has yet to be written. It would return the quantities Ak1, C, D, omegahat, Phi, etc. as slots, and then show.Coef.rrar would also need to be written.

Author(s)

T. W. Yee

References

Ahn, S. and Reinsel, G. C. (1988). Nested reduced-rank autoregressive models for multiple time series. *Journal of the American Statistical Association*, **83**, 849–856.

See Also

vglm, grain.us.

Examples

```
## Not run:
year <- seq(1961 + 1/12, 1972 + 10/12, by = 1/12)
par(mar = c(4, 4, 2, 2) + 0.1, mfrow = c(2, 2))
for (ii in 1:4) {
  plot(year, grain.us[, ii], main = names(grain.us)[ii], las = 1,
       type = "l", xlab = "", ylab = "", col = "blue")
  points(year, grain.us[, ii], pch = "*", col = "blue")
}
apply(grain.us, 2, mean)  # mu vector
cgrain <- scale(grain.us, scale = FALSE)  # Center the time series only
fit <- vglm(cgrain ~ 1, rrar(Ranks = c(4, 1)), trace = TRUE)
summary(fit)

print(fit@misc$Ak1, digits = 2)
print(fit@misc$Cmatrices, digits = 3)
print(fit@misc$Dmatrices, digits = 3)
print(fit@misc$omegahat, digits = 3)
print(fit@misc$Phimatrices, digits = 2)

par(mar = c(4, 4, 2, 2) + 0.1, mfrow = c(4, 1))
for (ii in 1:4) {
  plot(year, fit@misc$Z[, ii], main = paste("Z", ii, sep = ""),
       type = "l", xlab = "", ylab = "", las = 1, col = "blue")
  points(year, fit@misc$Z[, ii], pch = "*", col = "blue")
}

## End(Not run)
```

---

rrvglm *Fitting Reduced-Rank Vector Generalized Linear Models (RR-VGLMs) and Doubly Constrained RR-VGLMs (DRR-VGLMs)*

Description

A *reduced-rank vector generalized linear model* (RR-VGLM) is fitted. RR-VGLMs are VGLMs but some of the constraint matrices are estimated. *Doubly constrained* RR-VGLMs (DRR-VGLMs) can also be fitted.

Usage

```
rrvglm(formula, family = stop("'family' is unassigned"),
       data = list(), weights = NULL, subset = NULL,
       na.action = na.fail, etastart = NULL, mustart = NULL,
       coefstart = NULL, control = rrvglm.control(...),
       offset = NULL, method = "rrvglm.fit", model = FALSE,
       x.arg = TRUE, y.arg = TRUE, contrasts = NULL,
       constraints = NULL, extra = NULL, qr.arg = FALSE,
       smart = TRUE, ...)
```

Arguments

`formula, family, weights`
: See `vglm`.

`data`
: an optional data frame containing the variables in the model. By default the variables are taken from `environment(formula)`, typically the environment from which `rrvglm` is called.

`subset, na.action`
: See `vglm`.

`etastart, mustart, coefstart`
: See `vglm`.

`control`
: a list of parameters for controlling the fitting process. See `rrvglm.control` for details.

`offset, model, contrasts`
: See `vglm`.

`method`
: the method to be used in fitting the model. The default (and presently only) method `rrvglm.fit` uses iteratively reweighted least squares (IRLS).

`x.arg, y.arg`
: logical values indicating whether the model matrix and response vector/matrix used in the fitting process should be assigned in the `x` and `y` slots. Note the model matrix is the LM model matrix; to get the VGLM model matrix type `model.matrix(vglmfit)` where `vglmfit` is a `vglm` object.

`constraints`
: See `vglm`.

`extra, smart, qr.arg`
: See `vglm`.

`...`
: further arguments passed into `rrvglm.control`.

Details

In this documentation, M is the number of linear predictors. For RR-VGLMs, the central formula is given by

$$\eta = B_1^T x_1 + A\nu$$

where x_1 is a vector (usually just a 1 for an intercept), x_2 is another vector of explanatory variables, and $\nu = C^T x_2$ is an R-vector of latent variables. Here, η is a vector of linear predictors, e.g., the mth element is $\eta_m = \log(E[Y_m])$ for the mth Poisson response. The dimension of η is M by definition. The matrices B_1, A and C are estimated from the data, i.e., contain the regression coefficients. For ecologists, the central formula represents a *constrained linear ordination* (CLO) since it is linear in the latent variables. It means that the response is a monotonically increasing or decreasing function of the latent variables.

For identifiability it is common to enforce *corner constraints* on A: by default, the top R by R submatrix is fixed to be the order-R identity matrix and the remainder of A is estimated.

The underlying algorithm of RR-VGLMs is iteratively reweighted least squares (IRLS) with an optimizing algorithm applied within each IRLS iteration (e.g., alternating algorithm).

In theory, any **VGAM** family function that works for `vglm` and `vgam` should work for `rrvglm` too. The function that actually does the work is `rrvglm.fit`; it is `vglm.fit` with some extra code.

Value

An object of class `"rrvglm"`, which has the the same slots as a `"vglm"` object. The only difference is that the some of the constraint matrices are estimates rather than known. But **VGAM** stores the models the same internally. The slots of `"vglm"` objects are described in `vglm-class`.

Note

The arguments of `rrvglm` are in general the same as those of `vglm` but with some extras in `rrvglm.control`.

The smart prediction (`smartpred`) library is packed with the **VGAM** library.

In an example below, a rank-1 *stereotype* (reduced-rank multinomial logit) model of Anderson (1984) is fitted to some car data. The reduced-rank regression is performed, adjusting for two covariates. Setting a trivial constraint matrix (`diag(M)`) for the latent variable variables in x_2 avoids a warning message when it is overwritten by a (common) estimated constraint matrix. It shows that German cars tend to be more expensive than American cars, given a car of fixed weight and width.

If `fit <- rrvglm(..., data = mydata)` then `summary(fit)` requires corner constraints and no missing values in `mydata`. Often the estimated variance-covariance matrix of the parameters is not positive-definite; if this occurs, try refitting the model with a different value for `Index.corner`.

For *constrained quadratic ordination* (CQO) see `cqo` for more details about QRR-VGLMs.

With multiple binary responses, one must use `binomialff(multiple.responses = TRUE)` to indicate that the response is a matrix with one response per column. Otherwise, it is interpreted as a single binary response variable.

To fit DRR-VGLMs see the arguments `H.A` and `H.C` in `rrvglm.control`. DRR-VGLMs provide structure to the **A** and **C** matrices via constraint matrices. So instead of them being general unstructured matrices, one can make many of their elements to have the same value, else be identically equal to 0, for example. This gives greater control over what is modelled as a latent variable, e.g., if one subset of the covariates are physical variables and the remainder are psychological variables then a rank-2 model might have each latent variable a linear combination of each of the types of variables separately.

Incidentally, before I forget, if `Corner = TRUE` then the `@H.A` slot indicates that the `Index.corner` rows of **A** are estimated. This is a remnant of some internal computations because it is more efficient to estimate the entire **A**, bar rows `str0`, and then normalize it. In contrast, optimizing over a subset of **A** is slow.

Author(s)

Thomas W. Yee

References

Yee, T. W. and Hastie, T. J. (2003). Reduced-rank vector generalized linear models. *Statistical Modelling*, **3**, 15–41.

Yee, T. W. (2004). A new technique for maximum-likelihood canonical Gaussian ordination. *Ecological Monographs*, **74**, 685–701.

Anderson, J. A. (1984). Regression and ordered categorical variables. *Journal of the Royal Statistical Society, Series B, Methodological*, **46**, 1–30.

Yee, T. W. (2014). Reduced-rank vector generalized linear models with two linear predictors. *Computational Statistics and Data Analysis*, **71**, 889–902.

Yee, T. W., Frigau, L. and Ma, C. (2024). Heaping and seeping, GAITD regression and doubly constrained reduced rank vector generalized linear models, in smoking studies. *In preparation*.

See Also

`rrvglm.control`, `summary.drrvglm`, `lvplot.rrvglm` (same as `biplot.rrvglm`), `rrvglm-class`, `grc`, `cqo`, `vglmff-class`, `vglm`, `vglm-class`, `smartpred`, `rrvglm.fit`. Special family functions include `negbinomial` `zipoisson` and `zinegbinomial`. (see Yee (2014) and what was formerly in **COZIGAM**). Methods functions include `Coef.rrvglm`, `calibrate.rrvglm`, etc. Data include `crashi`.

Examples

```
## Not run:
# Example 1: RR NB with Var(Y) = mu + delta1 * mu^delta2
nn <- 1000       # Number of observations
delta1 <- 3.0    # Specify this
delta2 <- 1.5    # Specify this; should be greater than 1
a21 <- 2 - delta2
mydata <- data.frame(x2 = runif(nn), x3 = runif(nn))
mydata <- transform(mydata, mu = exp(2 + 3 * x2 + 0 * x3))
mydata <- transform(mydata,
    y2 = rnbinom(nn, mu = mu, size = (1/delta1)*mu^a21))
plot(y2 ~ x2, mydata, pch = "+", col = 'blue', las = 1,
  main = paste0("Var(Y) = mu + ", delta1, " * mu^", delta2))
rrnb2 <- rrvglm(y2 ~ x2 + x3, negbinomial(zero = NULL),
                data = mydata, trace = TRUE)
```

```
a21.hat <- (Coef(rrnb2)@A)["loglink(size)", 1]
beta11.hat <- Coef(rrnb2)@B1["(Intercept)", "loglink(mu)"]
beta21.hat <- Coef(rrnb2)@B1["(Intercept)", "loglink(size)"]
(delta1.hat <- exp(a21.hat * beta11.hat - beta21.hat))
(delta2.hat <- 2 - a21.hat)
# delta1.hat:
# exp(a21.hat * predict(rrnb2)[1,1] - predict(rrnb2)[1,2])
summary(rrnb2)

# Obtain a 95 percent CI for delta2:
se.a21.hat <- sqrt(vcov(rrnb2)["I(latvar.mat)", "I(latvar.mat)"])
ci.a21 <- a21.hat +  c(-1, 1) * 1.96 * se.a21.hat
(ci.delta2 <- 2 - rev(ci.a21))  # The 95 percent CI

Confint.rrnb(rrnb2)  # Quick way to get it

# Plot the abundances and fitted values vs the latent variable
plot(y2 ~ latvar(rrnb2), data = mydata, col = "blue",
     xlab = "Latent variable", las = 1)
ooo <- order(latvar(rrnb2))
lines(fitted(rrnb2)[ooo] ~ latvar(rrnb2)[ooo], col = "red")

# Example 2: stereotype model (RR multinomial logit model)
data(car.all)
scar <- subset(car.all,
    is.element(Country, c("Germany", "USA", "Japan", "Korea")))
fcols <- c(13,14,18:20,22:26,29:31,33,34,36)  # These are factors
scar[, -fcols] <- scale(scar[, -fcols])  # Stdze all numerical vars
ones <- matrix(1, 3, 1)
clist <- list("(Intercept)" = diag(3), Width = ones, Weight = ones,
              Disp. = diag(3), Tank = diag(3), Price = diag(3),
              Frt.Leg.Room = diag(3))
set.seed(111)
fit <- rrvglm(Country ~ Width + Weight + Disp. + Tank +
              Price + Frt.Leg.Room,
              multinomial, data = scar, Rank = 2, trace = TRUE,
              constraints = clist, noRRR = ~ 1 + Width + Weight,
#             Uncor = TRUE, Corner = FALSE,  # orig.
              Index.corner = c(1, 3),  # Less correlation
              Bestof = 3)
fit@misc$deviance  # A history of the fits
Coef(fit)
biplot(fit, chull = TRUE, scores = TRUE, clty = 2, Ccex = 2,
       ccol = "blue", scol = "orange", Ccol = "darkgreen",
       Clwd = 2, main = "1=Germany, 2=Japan, 3=Korea, 4=USA")

## End(Not run)
```

`rrvglm-class` *Class "rrvglm"*

Description

Reduced-rank vector generalized linear models.

Objects from the Class

Objects can be created by calls to `rrvglm`.

Slots

`extra`: Object of class `"list"`; the `extra` argument on entry to `vglm`. This contains any extra information that might be needed by the family function.

`family`: Object of class `"vglmff"`. The family function.

`iter`: Object of class `"numeric"`. The number of IRLS iterations used.

`predictors`: Object of class `"matrix"` with M columns which holds the M linear predictors.

`assign`: Object of class `"list"`, from class `"vlm"`. This named list gives information matching the columns and the (LM) model matrix terms.

`call`: Object of class `"call"`, from class `"vlm"`. The matched call.

`coefficients`: Object of class `"numeric"`, from class `"vlm"`. A named vector of coefficients.

`constraints`: Object of class `"list"`, from class `"vlm"`. A named list of constraint matrices used in the fitting.

`contrasts`: Object of class `"list"`, from class `"vlm"`. The contrasts used (if any).

`control`: Object of class `"list"`, from class `"vlm"`. A list of parameters for controlling the fitting process. See `vglm.control` for details.

`criterion`: Object of class `"list"`, from class `"vlm"`. List of convergence criterion evaluated at the final IRLS iteration.

`df.residual`: Object of class `"numeric"`, from class `"vlm"`. The residual degrees of freedom.

`df.total`: Object of class `"numeric"`, from class `"vlm"`. The total degrees of freedom.

`dispersion`: Object of class `"numeric"`, from class `"vlm"`. The scaling parameter.

`effects`: Object of class `"numeric"`, from class `"vlm"`. The effects.

`fitted.values`: Object of class `"matrix"`, from class `"vlm"`. The fitted values. This is usually the mean but may be quantiles, or the location parameter, e.g., in the Cauchy model.

`misc`: Object of class `"list"`, from class `"vlm"`. A named list to hold miscellaneous parameters.

`model`: Object of class `"data.frame"`, from class `"vlm"`. The model frame.

`na.action`: Object of class `"list"`, from class `"vlm"`. A list holding information about missing values.

`offset`: Object of class `"matrix"`, from class `"vlm"`. If non-zero, a M-column matrix of offsets.

`post`: Object of class `"list"`, from class `"vlm"` where post-analysis results may be put.

`preplot`: Object of class `"list"`, from class `"vlm"` used by `plotvgam`; the plotting parameters may be put here.

`prior.weights`: Object of class `"matrix"`, from class `"vlm"` holding the initially supplied weights.

`qr`: Object of class `"list"`, from class `"vlm"`. QR decomposition at the final iteration.

`R`: Object of class "matrix", from class "vlm". The **R** matrix in the QR decomposition used in the fitting.

`rank`: Object of class "integer", from class "vlm". Numerical rank of the fitted model.

`residuals`: Object of class "matrix", from class "vlm". The *working* residuals at the final IRLS iteration.

`ResSS`: Object of class "numeric", from class "vlm". Residual sum of squares at the final IRLS iteration with the adjusted dependent vectors and weight matrices.

`smart.prediction`: Object of class "list", from class "vlm". A list of data-dependent parameters (if any) that are used by smart prediction.

`terms`: Object of class "list", from class "vlm". The `terms` object used.

`weights`: Object of class "matrix", from class "vlm". The weight matrices at the final IRLS iteration. This is in matrix-band form.

`x`: Object of class "matrix", from class "vlm". The model matrix (LM, not VGLM).

`xlevels`: Object of class "list", from class "vlm". The levels of the factors, if any, used in fitting.

`y`: Object of class "matrix", from class "vlm". The response, in matrix form.

`Xm2`: Object of class "matrix", from class "vlm". See `vglm-class`.

`Ym2`: Object of class "matrix", from class "vlm". See `vglm-class`.

`callXm2`: Object of class "call", from class "vlm". The matched call for argument `form2`.

`A.est`, `C.est`: Object of class "matrix". The estimates of **A** and **C**.

Extends

Class "vglm", directly. Class "vlm", by class "vglm".

Methods

biplot `signature(x = "rrvglm")`: biplot.

Coef `signature(object = "rrvglm")`: more detailed coefficients giving **A**, $\boldsymbol{B}_1$, **C**, etc.

biplot `signature(object = "rrvglm")`: biplot.

print `signature(x = "rrvglm")`: short summary of the object.

summary `signature(object = "rrvglm")`: a more detailed summary of the object.

Note

Two new slots for "rrvglm" were added compared to "vglm" objects, for **VGAM** 1.1-10. They are `A.est` and `C.est`.

Author(s)

Thomas W. Yee

References

Yee, T. W. and Hastie, T. J. (2003). Reduced-rank vector generalized linear models. *Statistical Modelling*, **3**, 15–41.

See Also

rrvglm, lvplot.rrvglm, vglmff-class.

Examples

```
## Not run:  # Rank-1 stereotype model of Anderson (1984)
pneumo <- transform(pneumo, let = log(exposure.time),
                            x3  = runif(nrow(pneumo)))  # Noise
fit <- rrvglm(cbind(normal, mild, severe) ~ let + x3,
              multinomial, data = pneumo, Rank = 1)
Coef(fit)

## End(Not run)
```

---

rrvglm.control   *Control Function for rrvglm()*

Description

Algorithmic constants and parameters for running rrvglm are set using this function. Doubly constrained RR-VGLMs (DRR-VGLMs) are also catered for.

Usage

```
rrvglm.control(Rank = 1, Algorithm = "alternating",
    Corner = TRUE, Uncorrelated.latvar = FALSE, Wmat = NULL,
    Svd.arg = FALSE,
    Index.corner = head(setdiff(seq(length(str0) + Rank), str0), Rank),
    Ainit = NULL, Alpha = 0.5, Bestof = 1, Cinit = NULL,
    Etamat.colmax = 10, sd.Ainit = 0.02, sd.Cinit = 0.02,
    str0 = NULL, noRRR = ~1, Norrr = NA, noWarning = FALSE,
    trace = FALSE, Use.Init.Poisson.QO = FALSE,
    checkwz = TRUE, Check.rank = TRUE, Check.cm.rank = TRUE,
    wzepsilon = .Machine$double.eps^0.75,
    H.A.alt = list(), H.C = list(), scaleA = FALSE,
    Crow1positive = TRUE, ...)
```

Arguments

Rank	The numerical rank R of the model. Must be an element from the set $\{1,2,\ldots,\min(M,p2)\}$. Here, the vector of explanatory variables **x** is partitioned into (**x1**,**x2**), which is of dimension *p1*+*p2*. The variables making up **x1** are given by the terms in noRRR argument, and the rest of the terms comprise **x2**.
Algorithm	Character string indicating what algorithm is to be used. The default is the first one. The choice "derivative" has been withdrawn in **VGAM** 1.1-10.

`Corner` Logical indicating whether corner constraints are to be used. This is one method for ensuring a unique solution. If `TRUE`, `Index.corner` specifies the R rows of the constraint matrices that are use as the corner constraints, i.e., they hold an order-R identity matrix.

`Uncorrelated.latvar` Logical indicating whether uncorrelated latent variables are to be used. This is normalization forces the variance-covariance matrix of the latent variables to be `diag(Rank)`, i.e., unit variance and uncorrelated. This constraint does not lead to a unique solution because it can be rotated. **Update during 2023/2024:** setting this argument to be `TRUE` might not work anymore.

`Wmat` Yet to be done.

`Svd.arg` Logical indicating whether a singular value decomposition of the outer product is to computed. This is another normalization which ensures uniqueness. See the argument `Alpha` below. **Update during 2023/2024:** setting this argument to be `TRUE` might not work anymore.

`Index.corner` Specifies the R rows of the constraint matrices that are used for the corner constraints, i.e., they hold an order-R identity matrix.

For certain DRR-VGLMs one does not want to have corner constraints (e.g., `CM.qnorm`) so setting (`scaleA = TRUE` and) `Corner = FALSE` will achieve this. Then arguments such as `Index.corner` and `str0` will be ignored. If there are structural zeros then they should be built into the constraint matrices.

`Alpha` The exponent in the singular value decomposition that is used in the first part: if the SVD is UDV^T then the first and second parts are UD^{α} and $D^{1-\alpha}V^T$ respectively. A value of 0.5 is 'symmetrical'. This argument is used only when `Svd.arg=TRUE`. **Update during 2023/2024:** using this argument might not work anymore.

`Bestof` Integer. The best of `Bestof` models fitted is returned. This argument helps guard against local solutions by (hopefully) finding the global solution from many fits. The argument works only when the function generates its own initial value for **C**, i.e., when **C** is *not* passed in as initial values.

`Ainit, Cinit` Initial **A** and **C** matrices which may speed up convergence. They must be of the correct dimension.

`Etamat.colmax` Positive integer, no smaller than `Rank`. Controls the amount of memory used by `.Init.Poisson.QO()`. It is the maximum number of columns allowed for the pseudo-response and its weights. In general, the larger the value, the better the initial value. Used only if `Use.Init.Poisson.QO=TRUE`.

`str0` Integer vector specifying which rows of the estimated constraint matrices (**A**) are to be all zeros. These are called *structural zeros*. Must not have any common value with `Index.corner`, and be a subset of the vector `1:M`. The default, `str0 = NULL`, means no structural zero rows at all.

`sd.Ainit, sd.Cinit` Standard deviation of the initial values for the elements of **A** and **C**. These are normally distributed with mean zero. This argument is used only if `Use.Init.Poisson.QO = FALSE`.

`noRRR` Formula giving terms that are *not* to be included in the reduced-rank regression. That is, `noRRR` specifes which explanatory variables are in the x_1 vector of `rrvglm`, and the rest go into x_2. The x_1 variables constitute the $\boldsymbol{B}_1$ matrix in Yee and Hastie (2003). Those x_2 variables which are subject to the reduced-rank regression correspond to the $\boldsymbol{B}_2$ matrix. Set `noRRR = NULL` for the reduced-rank regression to be applied to every explanatory variable including the intercept.

`Norrr` Defunct. Please use `noRRR`. Its use results in an error. The argument may be removed soon.

`trace` Logical indicating if output should be produced for each iteration.

`Use.Init.Poisson.QO`
: Logical indicating whether the `.Init.Poisson.QO()` should be used to obtain initial values for the **C**. The function uses a new method that can work well if the data are Poisson counts coming from an equal-tolerances QRR-VGLM (CQO). This option is less realistic for RR-VGLMs compared to QRR-VGLMs.

`checkwz` logical indicating whether the diagonal elements of the working weight matrices should be checked whether they are sufficiently positive, i.e., greater than `wzepsilon`. If not, any values less than `wzepsilon` are replaced with this value.

`noWarning, Check.rank, Check.cm.rank`
: Same as `vglm.control`. Ignored for **VGAM** 0.9-7 and higher.

`wzepsilon` Small positive number used to test whether the diagonals of the working weight matrices are sufficiently positive.

`H.A.alt, H.C` Lists. DRR-VGLMs are *doubly constrained* RR-VGLMs where **A** has `Rank` constraint matrices (one for each column) in a list called `H.A.alt`, and **C** has a constraint matrix for each row, i.e., for each explanatory variable making up it. The class `"drrvglm"` may arise for such models. So `H.C` should be a named list of p_2 constraint matrices, each one for a different row of **C**, i.e., p_2 is the number of variables making up the latent variable. Note that if `H.C` has `names` then matching is done with that, and the components of `H.C` are reordered if they are not sorted according to the terms in `formula`. If they are not named, then their *order* is used, for example, `H.C[[1]]` and `H.C[[2]]` are taken as the constraint matrices for the first two variables of the latent variable(s).

`scaleA` Logical. Another uniqueness constraint to obtain a unique **A** and **C**. If `H.A.alt` and/or `H.C` are inputted then sometimes one wants to preserve the structure in **A**, e.g., `CM.qnorm`. Here, `A <- scale(A, center = FALSE)` so that only the columns are multiplicatively scaled. Note that the estimates of the elements of **A** and **C** are unique, up to their sign. Also note that ideally the attributes `attr(,"scaled:scale")` should be unity upon convergence so that if they differ substantially from that then this suggests some misbehaviour in convergence.

`Crow1positive` Logical vector of length `Rank` (recycled if necessary): are the elements of the first row of C positive? For example, if `Rank` is 4, then specifying `Crow1positive = c(FALSE, TRUE)` will force $C[1,1]$ and $C[1,3]$ to be negative, and $C[1,2]$ and $C[1,4]$ to be positive. This argument therefore allows the user to determine the direction of the latent variables since they are unique up to a sign. This argument certainly works for RR-VGLMs but may not be applicable to DRR-VGLMs because the constraint matrices may control their sign.

`...` Variables in ... are passed into `vglm.control`. If the derivative algorithm is used then ... are also passed into `rrvglm.optim.control`; and if the alternating algorithm is used then ... are also passed into `valt0.control`.

In the above, R is the `Rank` and M is the number of linear predictors.

Details

VGAM supports three normalizations to ensure a unique solution. Of these, only corner constraints will work with summary of RR-VGLM objects. **Update during late-2023/early-2024:** with ongoing work implementing the "drrvglm" class, there may be disruption and changes to other normalizations. However, corner constraints should be fully supported and have the greatest priority.

Value

A list with components matching the input names. Some error checking is done, but not much.

Note

The arguments in this function begin with an upper case letter to help avoid interference with those of vglm.control.

In the example below a rank-1 *stereotype* model (Anderson, 1984) is fitted. However, the intercepts ideally should be sorted and that might now be achieved using CM.symm0, CM.equid, CM.qnorm, etc. Currently the intercepts are completely unconstrained.

Author(s)

Thomas W. Yee

References

Yee, T. W. and Hastie, T. J. (2003). Reduced-rank vector generalized linear models. *Statistical Modelling*, **3**, 15–41.

See Also

rrvglm, rrvglm-class, summary.drrvglm, rrvglm.optim.control, vglm, vglm.control, TypicalVGAMfamilyFunction, CM.qnorm, cqo.

Examples

```
## Not run:
set.seed(111)
pneumo <- transform(pneumo, let = log(exposure.time),
                            x3 = runif(nrow(pneumo)))  # Unrelated
fit <- rrvglm(cbind(normal, mild, severe) ~ let + x3,
              multinomial, pneumo, Rank = 1, Index.corner = 2)
constraints(fit)
vcov(fit)
summary(fit)

## End(Not run)
```

rrvglm.optim.control *Control Function for rrvglm() Calling optim()*

Description

Algorithmic constants and parameters for running optim within rrvglm are set using this function.

Usage

```
rrvglm.optim.control(Fnscale = 1, Maxit = 100,
                     Switch.optimizer = 3, Abstol = -Inf,
                     Reltol = sqrt(.Machine$double.eps), ...)
```

Arguments

Fnscale	Passed into optim as fnscale.
Maxit	Passed into optim as maxit.
Switch.optimizer	Iteration number when the "Nelder-Mead" method of optim is switched to the quasi-Newton "BFGS" method. Assigning Switch.optimizer a negative number means always BFGS, while assigning Switch.optimizer a value greater than maxits means always use Nelder-Mead.
Abstol	Passed into optim as abstol.
Reltol	Passed into optim as reltol.
...	Ignored.

Details

See optim for more details.

Value

A list with components equal to the arguments.

Note

The transition between optimization methods may be unstable, so users may have to vary the value of Switch.optimizer.

Practical experience with Switch.optimizer shows that setting it to too large a value may lead to a local solution, whereas setting it to a low value will obtain the global solution. It appears that, if BFGS kicks in too late when the Nelder-Mead algorithm is starting to converge to a local solution, then switching to BFGS will not be sufficient to bypass convergence to that local solution.

Author(s)

Thomas W. Yee

See Also

rrvglm.control, optim.

---

ruge *Rutherford-Geiger Polonium Data*

Description

Decay counts of polonium recorded by Rutherford and Geiger (1910).

Usage

```
data(ruge)
```

Format

This data frame contains the following columns:

counts a numeric vector, counts or frequencies

number a numeric vector, the number of decays

Details

These are the radioactive decay counts of polonium recorded by Rutherford and Geiger (1910) representing the number of scintillations in 2608 1/8 minute intervals. For example, there were 57 frequencies of zero counts. The counts can be thought of as being approximately Poisson distributed.

Source

Rutherford, E. and Geiger, H. (1910) The Probability Variations in the Distribution of alpha Particles, *Philosophical Magazine*, **20**, 698–704.

Examples

```
lambdahat <- with(ruge, weighted.mean(number, w = counts))
(N <- with(ruge, sum(counts)))
with(ruge, cbind(number, counts,
                 fitted = round(N * dpois(number, lambdahat))))
```

s *Defining Smooths in VGAM Formulas*

Description

s is used in the definition of (vector) smooth terms within vgam formulas. This corresponds to 1st-generation VGAMs that use backfitting for their estimation. The effective degrees of freedom is prespecified.

Usage

```
s(x, df = 4, spar = 0, ...)
```

Arguments

x	covariate (abscissae) to be smoothed. Note that x must be a *single* variable and not a function of a variable. For example, s(x) is fine but s(log(x)) will fail. In this case, let logx <- log(x) (in the data frame), say, and then use s(logx). At this stage bivariate smoothers (x would be a two-column matrix) are not implemented.
df	numerical vector of length r. Effective degrees of freedom: must lie between 1 (linear fit) and n (interpolation). Thus one could say that df-1 is the *effective nonlinear degrees of freedom* (ENDF) of the smooth. Recycling of values will be used if df is not of length r. If spar is positive then this argument is ignored. Thus s() means that the effective degrees of freedom is prespecified. If it is known that the component function(s) are more wiggly than usual then try increasing the value of this argument.
spar	numerical vector of length r. Positive smoothing parameters (after scaling) . Larger values mean more smoothing so that the solution approaches a linear fit for that component function. A zero value means that df is used. Recycling of values will be used if spar is not of length r.
...	Ignored for now.

Details

In this help file M is the number of additive predictors and r is the number of component functions to be estimated (so that r is an element from the set {1,2,...,M}). Also, if n is the number of *distinct* abscissae, then s will fail if $n < 7$.

s, which is symbolic and does not perform any smoothing itself, only handles a single covariate. Note that s works in vgam only. It has no effect in vglm (actually, it is similar to the identity function I so that s(x2) is the same as x2 in the LM model matrix). It differs from the s() of the **gam** package and the s of the **mgcv** package; they should not be mixed together. Also, terms involving s should be simple additive terms, and not involving interactions and nesting etc. For example, myfactor:s(x2) is not a good idea.

Value

A vector with attributes that are (only) used by vgam.

Note

The vector cubic smoothing spline which s() represents is computationally demanding for large M. The cost is approximately $O(nM^3)$ where n is the number of unique abscissae.

Currently a bug relating to the use of s() is that only constraint matrices whose columns are orthogonal are handled correctly. If any s() term has a constraint matrix that does not satisfy this condition then a warning is issued. See is.buggy for more information.

A more modern alternative to using s with vgam is to use sm.os or sm.ps. This does not require backfitting and allows automatic smoothing parameter selection. However, this alternative should only be used when the sample size is reasonably large (> 500, say). These are called Generation-2 VGAMs.

Another alternative to using s with vgam is bs and/or ns with vglm. The latter implements half-stepping, which is helpful if convergence is difficult.

Author(s)

Thomas W. Yee

References

Yee, T. W. and Wild, C. J. (1996). Vector generalized additive models. *Journal of the Royal Statistical Society, Series B, Methodological*, **58**, 481–493.

See Also

vgam, is.buggy, sm.os, sm.ps, vsmooth.spline.

Examples

```
# Nonparametric logistic regression
fit1 <- vgam(agaaus ~ s(altitude, df = 2), binomialff, data = hunua)
## Not run:  plot(fit1, se = TRUE)

# Bivariate logistic model with artificial data
nn <- 300
bdata <- data.frame(x1 = runif(nn), x2 = runif(nn))
bdata <- transform(bdata,
    y1 = rbinom(nn, size = 1, prob = logitlink(sin(2 * x2), inverse = TRUE)),
    y2 = rbinom(nn, size = 1, prob = logitlink(sin(2 * x2), inverse = TRUE)))
fit2 <- vgam(cbind(y1, y2) ~ x1 + s(x2, 3), trace = TRUE,
             binom2.or(exchangeable = TRUE), data = bdata)
coef(fit2, matrix = TRUE)  # Hard to interpret
## Not run:  plot(fit2, se = TRUE, which.term = 2, scol = "blue")
```

`sc.studentt2` *Scaled Student t Distribution with 2 df Family Function*

Description

Estimates the location and scale parameters of a scaled Student t distribution with 2 degrees of freedom, by maximum likelihood estimation.

Usage

```
sc.studentt2(percentile = 50, llocation = "identitylink",
    lscale = "loglink", ilocation = NULL, iscale = NULL,
    imethod = 1, zero = "scale")
```

Arguments

`percentile` A numerical vector containing values between 0 and 100, which are the quantiles and expectiles. They will be returned as 'fitted values'.

`llocation, lscale`
See `Links` for more choices, and `CommonVGAMffArguments`.

`ilocation, iscale, imethod, zero`
See `CommonVGAMffArguments` for details.

Details

Koenker (1993) solved for the distribution whose quantiles are equal to its expectiles. Its canonical form has mean and mode at 0, and has a heavy tail (in fact, its variance is infinite).

The standard ("canonical") form of this distribution can be endowed with a location and scale parameter. The standard form has a density that can be written as

$$f(z) = 2/(4 + z^2)^{3/2}$$

for real y. Then $z = (y - a)/b$ for location and scale parameters a and $b > 0$. The mean of Y is a. By default, $\eta_1 = a)$ and $\eta_2 = \log(b)$. The expectiles/quantiles corresponding to `percentile` are returned as the fitted values; in particular, `percentile = 50` corresponds to the mean (0.5 expectile) and median (0.5 quantile).

Note that if Y has a standard `dsc.t2` then $Y = \sqrt{2}T_2$ where T_2 has a Student-t distribution with 2 degrees of freedom. The two parameters here can also be estimated using `studentt2` by specifying `df = 2` and making an adjustment for the scale parameter, however, this **VGAM** family function is more efficient since the EIM is known (Fisher scoring is implemented.)

Value

An object of class "vglmff" (see `vglmff-class`). The object is used by modelling functions such as `vglm`, `rrvglm` and `vgam`.

Author(s)

T. W. Yee

References

Koenker, R. (1993). When are expectiles percentiles? (solution) *Econometric Theory*, **9**, 526–527.

See Also

`dsc.t2`, `studentt2`.

Examples

```
set.seed(123); nn <- 1000
kdata <- data.frame(x2 = sort(runif(nn)))
kdata <- transform(kdata, mylocat = 1 + 3 * x2,
                          myscale = 1)
kdata <- transform(kdata, y = rsc.t2(nn, loc = mylocat, scale = myscale))
fit  <- vglm(y ~ x2, sc.studentt2(perc = c(1, 50, 99)), data = kdata)
fit2 <- vglm(y ~ x2,    studentt2(df = 2), data = kdata)  # 'same' as fit

coef(fit, matrix = TRUE)
head(fitted(fit))
head(predict(fit))

# Nice plot of the results
## Not run:  plot(y ~ x2, data = kdata, col = "blue", las = 1,
     sub  = paste("n =", nn),
     main = "Fitted quantiles/expectiles using the sc.studentt2() distribution")
matplot(with(kdata, x2), fitted(fit), add = TRUE, type = "l", lwd = 3)
legend("bottomright", lty = 1:3, lwd = 3, legend = colnames(fitted(fit)),
       col = 1:3)
## End(Not run)

fit@extra$percentile  # Sample quantiles
```

`score.stat` *Rao's Score Test Statistics Evaluated at the Null Values*

Description

Generic function that computes Rao's score test statistics evaluated at the null values.

Usage

```
score.stat(object, ...)
score.stat.vlm(object, values0 = 0, subset = NULL, omit1s = TRUE,
           all.out = FALSE, orig.SE = FALSE, iterate.SE = TRUE,
           iterate.score = TRUE, trace = FALSE, ...)
```

Arguments

object, values0, subset
: Same as in wald.stat.vlm.

omit1s, all.out
: Same as in wald.stat.vlm.

orig.SE, iterate.SE
: Same as in wald.stat.vlm.

iterate.score
: Logical. The score vector is evaluated at one value of values0 and at other regression coefficient values. These other values may be either the MLE obtained from the original object (FALSE), else at values obtained by further IRLS iterations—this argument enables that choice.

trace
: Same as in wald.stat.vlm.

...
: Ignored for now.

Details

The (Rao) *score test* (also known as the *Lagrange multiplier test* in econometrics) is a third general method for hypothesis testing under a likelihood-based framework (the others are the likelihood ratio test and Wald test; see lrt.stat and wald.stat). Asymptotically, the three tests are equivalent. The Wald test is not invariant to parameterization, and the usual Wald test statistics computed at the estimates make it vulnerable to the Hauck-Donner effect (HDE; see hdeff). This function is similar to wald.stat in that one coefficient is set to 0 (by default) and the *other* coefficients are iterated by IRLS to get their MLE subject to this constraint. The SE is almost always based on the expected information matrix (EIM) rather than the OIM, and for some models the EIM and OIM coincide.

Value

By default the signed square root of the Rao score statistics are returned. If all.out = TRUE then a list is returned with the following components: score.stat the score statistic, SE0 the standard error of that coefficient, values0 the null values. Approximately, the default score statistics output are standard normal random variates if each null hypothesis is true.

Altogether, by the eight combinations of iterate.SE, iterate.score and orig.SE, there are six different variants of the Rao score statistic that can be returned because the score vector has 2 and the SEs have 3 subvariants.

Warning

See wald.stat.vlm.

Author(s)

Thomas W. Yee

See Also

wald.stat, lrt.stat, summaryvglm, summary.glm, anova.vglm, vglm, hdeff.

Examples

```
set.seed(1)
pneumo <- transform(pneumo, let = log(exposure.time),
                            x3 = rnorm(nrow(pneumo)))
(pfit <- vglm(cbind(normal, mild, severe) ~ let + x3, propodds, pneumo))
score.stat(pfit)  # No HDE here; should be similar to the next line:
coef(summary(pfit))[, "z value"]  # Wald statistics computed at the MLE
summary(pfit, score0 = TRUE)
```

`seglines` *Hauck-Donner Effects: Segmented Lines Plot*

Description

Plots the piecewise segmented curve made up of Wald statistics versus estimates, using a colour code for the HDE severity.

Usage

```
seglines(x, y, dy, ddy, lwd = 2, cex = 2, plot.it = TRUE,
   add.legend = TRUE, cex.legend = 1,
   position.legend = "topleft", eta0 = NA, COPS0 = NA,
   lty.table = c("solid", "dashed", "solid", "dashed",
                 "solid", "dashed", "solid"),
   col.table = rainbow.sky[-5], pch.table = 7:1,
   severity.table = c("None", "Faint", "Weak", "Moderate",
   "Strong", "Extreme", "Undetermined"), FYI = FALSE, ...)
```

Arguments

`x, y, dy, ddy` Same as `hdeffsev`.

`lwd, cex` Graphical parameters: line width, and character expansion.

`plot.it` Logical, plot it? If `FALSE` then the other graphical arguments are ignored.

`add.legend, position.legend`
Logical and character; add a legend? The other argument is fed into `legend`.

`cex.legend` Self-explanatory.

`severity.table, eta0, COPS0`
Same as `hdeffsev`.

`lty.table, col.table, pch.table`
Graphical parameters for the 7 different types of segments. Usually users should not assign anything to these arguments. Setting `pch.table = NULL` will suppress pch symbols from the legend.

`FYI, ...` Should be ignored.

Details

This function was written to complement hdeffsev and is rough-and-ready. It plots the signed Wald statistics as a function of the estimates, and uses a colour-code to indicate the severity of the Hauck-Donner effect (HDE). This can be obtained from its first two derivatives.

Value

This function returns the severity of the HDE, possibly invisibly.

Note

This function is likely to change in the short future because it is experimental and far from complete.

Author(s)

Thomas W. Yee.

See Also

hdeff, hdeffsev.

Examples

```
deg <- 4  # myfun is a function that approximates the HDE
myfun <- function(x, deriv = 0) switch(as.character(deriv),
  '0' = x^deg * exp(-x),
  '1' = (deg * x^(deg-1) - x^deg) * exp(-x),
  '2' = (deg * (deg-1) * x^(deg-2) - 2*deg * x^(deg-1) +
         x^deg) * exp(-x))
## Not run:
curve(myfun, 0, 10, col = "white")
xgrid <- seq(0, 10, length = 101)
seglines(xgrid, myfun(xgrid), myfun(xgrid, deriv = 1),
         COPS0 = 2,
         myfun(xgrid, deriv = 2), pch.table = NULL,
         position = "bottom")

## End(Not run)
```

Select *Select Variables for a Formula Response or the RHS of a Formula*

Description

Select variables from a data frame whose names begin with a certain character string.

Usage

```
Select(data = list(), prefix = "y",
       lhs = NULL, rhs = NULL, rhs2 = NULL, rhs3 = NULL,
       as.character = FALSE, as.formula.arg = FALSE, tilde = TRUE,
       exclude = NULL, sort.arg = TRUE)
```

Arguments

`data`	A data frame or a matrix.
`prefix`	A vector of character strings, or a logical. If a character then the variables chosen from `data` begin with the value of `prefix`. If a logical then only `TRUE` is accepted and all the variables in `data` are chosen.
`lhs`	A character string. The response of a formula.
`rhs`	A character string. Included as part of the RHS a formula. Set `rhs = "0"` to suppress the intercept.
`rhs2, rhs3`	Same as `rhs` but appended to its RHS, i.e., `paste0(rhs, " + ", rhs2, " + ", rhs3)`. If used, `rhs` should be used first, and then possibly `rhs2` and then possibly `rhs3`.
`as.character`	Logical. Return the answer as a character string?
`as.formula.arg`	Logical. Is the answer a formula?
`tilde`	Logical. If `as.character` and `as.formula.arg` are both `TRUE` then include the tilde in the formula?
`exclude`	Vector of character strings. Exclude these variables explicitly.
`sort.arg`	Logical. Sort the variables?

Details

This is meant as a utility function to avoid manually: (i) making a `cbind` call to construct a big matrix response, and (ii) constructing a formula involving a lot of terms. The savings can be made because the variables of interest begin with some prefix, e.g., with the character `"y"`.

Value

If `as.character = FALSE` and `as.formula.arg = FALSE` then a matrix such as `cbind(y1, y2, y3)`. If `as.character = TRUE` and `as.formula.arg = FALSE` then a character string such as `"cbind(y1, y2, y3)"`.

If `as.character = FALSE` and `as.formula.arg = TRUE` then a `formula` such as `lhs ~ y1 + y2 + y3`. If `as.character = TRUE` and `as.formula.arg = TRUE` then a character string such as `"lhs ~ y1 + y2 + y3"`. See the examples below. By default, if no variables beginning the the value of `prefix` is found then a `NULL` is returned. Setting `prefix = " "` is a way of selecting no variables.

Note

This function is a bit experimental at this stage and may change in the short future. Some of its utility may be better achieved using `subset` and its `select` argument, e.g., `subset(pdata, TRUE, select = y01:y10)`.

For some models such as posbernoulli.t the order of the variables in the xij argument is crucial, therefore care must be taken with the argument sort.arg. In some instances, it may be good to rename variables y1 to y01, y2 to y02, etc. when there are variables such as y14.

Currently subsetcol() and Select() are identical. One of these functions might be withdrawn in the future.

Author(s)

T. W. Yee.

See Also

vglm, cbind, subset, formula, fill1.

Examples

```
Pneumo <- pneumo
colnames(Pneumo) <- c("y1", "y2", "y3", "x2")  # The "y" variables are response
Pneumo$x1 <- 1; Pneumo$x3 <- 3; Pneumo$x <- 0; Pneumo$x4 <- 4  # Add these

Select(data = Pneumo)  # Same as with(Pneumo, cbind(y1, y2, y3))
Select(Pneumo, "x")
Select(Pneumo, "x", sort = FALSE, as.char = TRUE)
Select(Pneumo, "x", exclude = "x1")
Select(Pneumo, "x", exclude = "x1", as.char = TRUE)
Select(Pneumo, c("x", "y"))
Select(Pneumo, "z")  # Now returns a NULL
Select(Pneumo, " ")  # Now returns a NULL
Select(Pneumo, prefix = TRUE, as.formula = TRUE)
Select(Pneumo, "x", exclude = c("x3", "x1"), as.formula = TRUE,
       lhs = "cbind(y1, y2, y3)", rhs = "0")
Select(Pneumo, "x", exclude = "x1", as.formula = TRUE, as.char = TRUE,
       lhs = "cbind(y1, y2, y3)", rhs = "0")

# Now a 'real' example:
Huggins89table1 <- transform(Huggins89table1, x3.tij = t01)
tab1 <- subset(Huggins89table1,
               rowSums(Select(Huggins89table1, "y")) > 0)
# Same as
# subset(Huggins89table1, y1 + y2 + y3 + y4 + y5 + y6 + y7 + y8 + y9 + y10 > 0)

# Long way to do it:
fit.th <-
   vglm(cbind(y01, y02, y03, y04, y05, y06, y07, y08, y09, y10) ~ x2 + x3.tij,
        xij = list(x3.tij ~ t01 + t02 + t03 + t04 + t05 + t06 + t07 + t08 +
                            t09 + t10 - 1),
        posbernoulli.t(parallel.t = TRUE ~ x2 + x3.tij),
        data = tab1, trace = TRUE,
        form2 = ~ x2 + x3.tij + t01 + t02 + t03 + t04 + t05 + t06 + t07 + t08 +
                                t09 + t10)
# Short way to do it:
Fit.th <- vglm(Select(tab1, "y") ~ x2 + x3.tij,
```

```
          xij = list(Select(tab1, "t", as.formula = TRUE,
                            sort = FALSE, lhs = "x3.tij", rhs = "0")),
          posbernoulli.t(parallel.t = TRUE ~ x2 + x3.tij),
          data = tab1, trace = TRUE,
          form2 = Select(tab1, prefix = TRUE, as.formula = TRUE))
```

---

seq2binomial *The Two-stage Sequential Binomial Distribution Family Function*

Description

Estimation of the probabilities of a two-stage binomial distribution.

Usage

```
seq2binomial(lprob1 = "logitlink", lprob2 = "logitlink",
             iprob1 = NULL,    iprob2 = NULL,
             parallel = FALSE, zero = NULL)
```

Arguments

`lprob1, lprob2`	Parameter link functions applied to the two probabilities, called p and q below. See `Links` for more choices.
`iprob1, iprob2`	Optional initial value for the first and second probabilities respectively. A `NULL` means a value is obtained in the `initialize` slot.
`parallel, zero`	Details at `Links`. If `parallel = TRUE` then the constraint also applies to the intercept. See `CommonVGAMffArguments` for details.

Details

This **VGAM** family function fits the model described by Crowder and Sweeting (1989) which is described as follows. Each of m spores has a probability p of germinating. Of the y_1 spores that germinate, each has a probability q of bending in a particular direction. Let y_2 be the number that bend in the specified direction. The probability model for this data is $P(y_1, y_2) =$

$$\binom{m}{y_1} p^{y_1} (1-p)^{m-y_1} \binom{y_1}{y_2} q^{y_2} (1-q)^{y_1-y_2}$$

for $0 < p < 1$, $0 < q < 1$, $y_1 = 1, \ldots, m$ and $y_2 = 1, \ldots, y_1$. Here, p is `prob1`, q is `prob2`.

Although the Authors refer to this as the *bivariate binomial* model, I have named it the *(two-stage) sequential binomial* model. Fisher scoring is used.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Note

The response must be a two-column matrix of sample proportions corresponding to y_1 and y_2. The m values should be inputted with the `weights` argument of `vglm` and `vgam`. The fitted value is a two-column matrix of estimated probabilities p and q. A common form of error is when there are no trials for y_1, e.g., if `mvector` below has some values which are zero.

Author(s)

Thomas W. Yee

References

Crowder, M. and Sweeting, T. (1989). Bayesian inference for a bivariate binomial distribution. *Biometrika*, **76**, 599–603.

See Also

`binomialff`, `cfibrosis`.

Examples

```
sdata <- data.frame(mvector = round(rnorm(nn <- 100, m = 10, sd = 2)),
                    x2 = runif(nn))
sdata <- transform(sdata, prob1 = logitlink(+2 - x2, inverse = TRUE),
                          prob2 = logitlink(-2 + x2, inverse = TRUE))
sdata <- transform(sdata, successes1 = rbinom(nn, size = mvector,    prob = prob1))
sdata <- transform(sdata, successes2 = rbinom(nn, size = successes1, prob = prob2))
sdata <- transform(sdata, y1 = successes1 / mvector)
sdata <- transform(sdata, y2 = successes2 / successes1)
fit <- vglm(cbind(y1, y2) ~ x2, seq2binomial, weight = mvector,
            data = sdata, trace = TRUE)
coef(fit)
coef(fit, matrix = TRUE)
head(fitted(fit))
head(depvar(fit))
head(weights(fit, type = "prior"))  # Same as with(sdata, mvector)
# Number of first successes:
head(depvar(fit)[, 1] * c(weights(fit, type = "prior")))
# Number of second successes:
head(depvar(fit)[, 2] * c(weights(fit, type = "prior")) *
                          depvar(fit)[, 1])
```

`setup.smart` *Smart Prediction Setup*

Description

Sets up smart prediction in one of two modes: `"write"` and `"read"`.

Usage

```
setup.smart(mode.arg, smart.prediction = NULL, max.smart = 30)
```

Arguments

- `mode.arg` — `mode.arg` must be `"write"` or `"read"`. If in `"read"` mode then `smart.prediction` must be assigned the data structure `.smart.prediction` that was created while fitting. This is stored in `object@smart.prediction` or `object$smart.prediction` where `object` is the name of the fitted object.
- `smart.prediction` — If in `"read"` mode then `smart.prediction` must be assigned the list of data dependent parameters, which is stored on the fitted object. Otherwise, `smart.prediction` is ignored.
- `max.smart` — `max.smart` is the initial length of the list `.smart.prediction`. It is not important because `.smart.prediction` is made larger if needed.

Details

This function is only required by programmers writing a modelling function such as `lm` and `glm`, or a prediction functions of such, e.g., `predict.lm`. The function `setup.smart` operates by mimicking the operations of a first-in first-out stack (better known as a *queue*).

Value

Nothing is returned.

Side Effects

In `"write"` mode `.smart.prediction` in `smartpredenv` is assigned an empty list with `max.smart` components. In `"read"` mode `.smart.prediction` in `smartpredenv` is assigned `smart.prediction`. Then `.smart.prediction.counter` in `smartpredenv` is assigned the value 0, and `.smart.prediction.mode` and `.max.smart` are written to `smartpredenv` too.

See Also

`lm`, `predict.lm`.

Examples

```
## Not run:
setup.smart("write")  # Put at the beginning of lm

## End(Not run)

## Not run: # Put at the beginning of predict.lm
setup.smart("read", smart.prediction = object$smart.prediction)

## End(Not run)
```

Simplex *Simplex Distribution*

Description

Density function, and random generation for the simplex distribution.

Usage

```
dsimplex(x, mu = 0.5, dispersion = 1, log = FALSE)
rsimplex(n, mu = 0.5, dispersion = 1)
```

Arguments

x	Vector of quantiles. The support of the distribution is the interval $(0, 1)$.
mu, dispersion	Mean and dispersion parameters. The former lies in the interval $(0, 1)$ and the latter is positive.
n, log	Same usage as runif.

Details

The **VGAM** family function simplex fits this model; see that online help for more information. For rsimplex() the rejection method is used; it may be very slow if the density is highly peaked, and will fail if the density asymptotes at the boundary.

Value

dsimplex(x) gives the density function, rsimplex(n) gives n random variates.

Author(s)

T. W. Yee

See Also

simplex.

Examples

```
sigma <- c(4, 2, 1)  # Dispersion parameter
mymu  <- c(0.1, 0.5, 0.7); xxx <- seq(0, 1, len = 501)
## Not run:  par(mfrow = c(3, 3))  # Figure 2.1 of Song (2007)
for (iii in 1:3)
  for (jjj in 1:3) {
    plot(xxx, dsimplex(xxx, mymu[jjj], sigma[iii]),
         type = "l", col = "blue", xlab = "", ylab = "", main =
         paste("mu = ", mymu[jjj], ", sigma = ", sigma[iii], sep = "")) }
## End(Not run)
```

simplex *Simplex Distribution Family Function*

Description

The two parameters of the univariate standard simplex distribution are estimated by full maximum likelihood estimation.

Usage

```
simplex(lmu = "logitlink", lsigma = "loglink", imu = NULL, isigma = NULL,
        imethod = 1, ishrinkage = 0.95, zero = "sigma")
```

Arguments

`lmu, lsigma`	Link function for `mu` and `sigma`. See `Links` for more choices.
`imu, isigma`	Optional initial values for `mu` and `sigma`. A `NULL` means a value is obtained internally.
`imethod, ishrinkage, zero`	See `CommonVGAMffArguments` for information.

Details

The probability density function can be written

$$f(y;\mu,\sigma) = [2\pi\sigma^2(y(1-y))^3]^{-0.5}\exp[-0.5(y-\mu)^2/(\sigma^2 y(1-y)\mu^2(1-\mu)^2)]$$

for $0 < y < 1$, $0 < \mu < 1$, and $\sigma > 0$. The mean of Y is μ (called `mu`, and returned as the fitted values).

The second parameter, `sigma`, of this standard simplex distribution is known as the dispersion parameter. The unit variance function is $V(\mu) = \mu^3(1-\mu)^3$. Fisher scoring is applied to both parameters.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Note

This distribution is potentially useful for dispersion modelling. Numerical problems may occur when `mu` is very close to 0 or 1.

Author(s)

T. W. Yee

References

Jorgensen, B. (1997). *The Theory of Dispersion Models*. London: Chapman & Hall

Song, P. X.-K. (2007). *Correlated Data Analysis: Modeling, Analytics, and Applications*. Springer.

See Also

dsimplex, dirichlet, rig, binomialff.

Examples

```
sdata <- data.frame(x2 = runif(nn <- 1000))
sdata <- transform(sdata, eta1 = 1 + 2 * x2,
                          eta2 = 1 - 2 * x2)
sdata <- transform(sdata, y = rsimplex(nn, mu = logitlink(eta1, inverse = TRUE),
                                       dispersion = exp(eta2)))
(fit <- vglm(y ~ x2, simplex(zero = NULL), data = sdata, trace = TRUE))
coef(fit, matrix = TRUE)
summary(fit)
```

---

simulate.vlm *Simulate Responses for VGLMs and VGAMs*

Description

Simulate one or more responses from the distribution corresponding to a fitted model object.

Usage

```
## S3 method for class 'vlm'
simulate(object, nsim = 1, seed = NULL, ...)
```

Arguments

object	an object representing a fitted model. Usually an object of class vglm-class or vgam-class.
nsim, seed	Same as simulate.
...	additional optional arguments.

Details

This is a methods function for simulate and hopefully should behave in a very similar manner. Only **VGAM** family functions with a `simslot` slot have been implemented for simulate.

Value

Similar to simulate. Note that many **VGAM** family functions can handle multiple responses. This can result in a longer data frame with more rows (`nsim` multiplied by `n` rather than the ordinary `n`). In the future an argument may be available so that there is always `n` rows no matter how many responses were inputted.

Warning

With multiple response and/or multivariate responses, the order of the elements may differ. For some **VGAM** families, the order is $n \times N \times F$, where n is the sample size, N is `nsim` and F is `ncol(fitted(vglmObject))`. For other **VGAM** families, the order is $n \times F \times N$. An example of each is given below.

See Also

Currently the **VGAM** family functions with a `simslot` slot are: `alaplace1`, `alaplace2`, `betabinomial`, `betabinomialff`, `betaR`, `betaff`, `biamhcop`, `bifrankcop`, `bilogistic`, `binomialff`, `binormal`, `binormalcop`, `biclaytoncop`, `cauchy`, `cauchy1`, `chisq`, `dirichlet`, `dagum`, `erlang`, `exponential`, `bifgmcop`, `fisk`, `gamma1`, `gamma2`, `gammaR`, `gengamma.stacy`, `geometric`, `gompertz`, `gumbelII`, `hzeta`, `inv.lomax`, `inv.paralogistic`, `kumar`, `lgamma1`, `lgamma3`, `lindley`, `lino`, `logff`, `logistic1`, `logistic`, `lognormal`, `lomax`, `makeham`, `negbinomial`, `negbinomial.size`, `paralogistic`, `perks`, `poissonff`, `posnegbinomial`, `posnormal`, `pospoisson`, `polya`, `polyaR`, `posbinomial`, `rayleigh`, `riceff`, `simplex`, `sinmad`, `slash`, `studentt`, `studentt2`, `studentt3`, `triangle`, `uninormal`, `yulesimon`, `zageometric`, `zageometricff`, `zanegbinomial`, `zanegbinomialff`, `zapoisson`, `zapoissonff`, `zigeometric`, `zigeometricff`, `zinegbinomial`, `zipf`, `zipoisson`, `zipoissonff`.

See also `RNG` about random number generation in R, `vglm`, `vgam` for model fitting.

Examples

```
nn <- 10; mysize <- 20; set.seed(123)
bdata <- data.frame(x2 = rnorm(nn))
bdata <- transform(bdata,
  y1   = rbinom(nn, size = mysize, p = logitlink(1+x2, inverse = TRUE)),
  y2   = rbinom(nn, size = mysize, p = logitlink(1+x2, inverse = TRUE)),
  f1   = factor(as.numeric(rbinom(nn, size = 1,
                                  p = logitlink(1+x2, inverse = TRUE)))))
(fit1 <- vglm(cbind(y1, aaa = mysize - y1) ~ x2,  # Matrix response (2-colns)
              binomialff, data = bdata))
(fit2 <- vglm(f1 ~ x2, binomialff, model = TRUE, data = bdata)) # Factor response

set.seed(123); simulate(fit1, nsim = 8)
set.seed(123); c(simulate(fit2, nsim = 3))  # Use c() when model = TRUE

# An n x N x F example
set.seed(123); n <- 100
bdata <- data.frame(x2 = runif(n), x3 = runif(n))
bdata <- transform(bdata, y1 = rnorm(n, 1 + 2 * x2),
                          y2 = rnorm(n, 3 + 4 * x2))
fit1 <- vglm(cbind(y1, y2) ~ x2, binormal(eq.sd = TRUE), data = bdata)
nsim <- 1000  # Number of simulations for each observation
my.sims <- simulate(fit1, nsim = nsim)
dim(my.sims)  # A data frame
aaa <- array(unlist(my.sims), c(n, nsim, ncol(fitted(fit1))))  # n by N by F
summary(rowMeans(aaa[, , 1]) - fitted(fit1)[, 1])  # Should be all 0s
summary(rowMeans(aaa[, , 2]) - fitted(fit1)[, 2])  # Should be all 0s

# An n x F x N example
```

```
n <- 100; set.seed(111); nsim <- 1000
zdata <- data.frame(x2 = runif(n))
zdata <- transform(zdata, lambda1 =  loglink(-0.5 + 2 * x2, inverse = TRUE),
                          lambda2 =  loglink( 0.5 + 2 * x2, inverse = TRUE),
                          pstr01  = logitlink( 0,            inverse = TRUE),
                          pstr02  = logitlink(-1.0,          inverse = TRUE))
zdata <- transform(zdata, y1 = rzipois(n, lambda = lambda1, pstr0 = pstr01),
                          y2 = rzipois(n, lambda = lambda2, pstr0 = pstr02))
zip.fit  <- vglm(cbind(y1, y2) ~ x2, zipoissonff, data = zdata, crit = "coef")
my.sims <- simulate(zip.fit, nsim = nsim)
dim(my.sims)  # A data frame
aaa <- array(unlist(my.sims), c(n, ncol(fitted(zip.fit)), nsim))  # n by F by N
summary(rowMeans(aaa[, 1, ]) - fitted(zip.fit)[, 1])  # Should be all 0s
summary(rowMeans(aaa[, 2, ]) - fitted(zip.fit)[, 2])  # Should be all 0s
```

Sinmad *The Singh-Maddala Distribution*

Description

Density, distribution function, quantile function and random generation for the Singh-Maddala distribution with shape parameters a and q, and scale parameter scale.

Usage

```
dsinmad(x, scale = 1, shape1.a, shape3.q, log = FALSE)
psinmad(q, scale = 1, shape1.a, shape3.q, lower.tail = TRUE, log.p = FALSE)
qsinmad(p, scale = 1, shape1.a, shape3.q, lower.tail = TRUE, log.p = FALSE)
rsinmad(n, scale = 1, shape1.a, shape3.q)
```

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
n	number of observations. If length(n) > 1, the length is taken to be the number required.
shape1.a, shape3.q	shape parameters.
scale	scale parameter.
log	Logical. If log = TRUE then the logarithm of the density is returned.
lower.tail, log.p	Same meaning as in pnorm or qnorm.

Details

See sinmad, which is the **VGAM** family function for estimating the parameters by maximum likelihood estimation.

Value

`dsinmad` gives the density, `psinmad` gives the distribution function, `qsinmad` gives the quantile function, and `rsinmad` generates random deviates.

Note

The Singh-Maddala distribution is a special case of the 4-parameter generalized beta II distribution.

Author(s)

T. W. Yee and Kai Huang

References

Kleiber, C. and Kotz, S. (2003). *Statistical Size Distributions in Economics and Actuarial Sciences*, Hoboken, NJ, USA: Wiley-Interscience.

See Also

`sinmad`, `genbetaII`.

Examples

```
sdata <- data.frame(y = rsinmad(n = 3000, scale = exp(2),
                                shape1 = exp(1), shape3 = exp(1)))
fit <- vglm(y ~ 1, sinmad(lss = FALSE, ishape1.a = 2.1), data = sdata,
               trace = TRUE, crit = "coef")
coef(fit, matrix = TRUE)
Coef(fit)
```

`sinmad` *Singh-Maddala Distribution Family Function*

Description

Maximum likelihood estimation of the 3-parameter Singh-Maddala distribution.

Usage

```
sinmad(lscale = "loglink", lshape1.a = "loglink", lshape3.q = "loglink",
         iscale = NULL, ishape1.a = NULL, ishape3.q = NULL, imethod = 1,
         lss = TRUE, gscale = exp(-5:5), gshape1.a = exp(-5:5),
         gshape3.q = exp(-5:5), probs.y = c(0.25, 0.5, 0.75),
         zero = "shape")
```

Arguments

- `lss` See `CommonVGAMffArguments` for important information.
- `lshape1.a, lscale, lshape3.q` Parameter link functions applied to the (positive) parameters a, `scale`, and q. See `Links` for more choices.
- `iscale, ishape1.a, ishape3.q, imethod, zero` See `CommonVGAMffArguments` for information. For `imethod = 2` a good initial value for `ishape3.q` is needed to obtain good estimates for the other parameters.
- `gscale, gshape1.a, gshape3.q` See `CommonVGAMffArguments` for information.
- `probs.y` See `CommonVGAMffArguments` for information.

Details

The 3-parameter Singh-Maddala distribution is the 4-parameter generalized beta II distribution with shape parameter $p = 1$. It is known under various other names, such as the Burr XII (or just the Burr distribution), Pareto IV, beta-P, and generalized log-logistic distribution. More details can be found in Kleiber and Kotz (2003).

Some distributions which are special cases of the 3-parameter Singh-Maddala are the Lomax ($a = 1$), Fisk ($q = 1$), and paralogistic ($a = q$).

The Singh-Maddala distribution has density

$$f(y) = aqy^{a-1}/[b^a\{1 + (y/b)^a\}^{1+q}]$$

for $a > 0$, $b > 0$, $q > 0$, $y \geq 0$. Here, b is the scale parameter `scale`, and the others are shape parameters. The cumulative distribution function is

$$F(y) = 1 - [1 + (y/b)^a]^{-q}.$$

The mean is

$$E(Y) = b\,\Gamma(1 + 1/a)\,\Gamma(q - 1/a)/\Gamma(q)$$

provided $-a < 1 < aq$; these are returned as the fitted values. This family function handles multiple responses.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Note

See the notes in `genbetaII`.

Author(s)

T. W. Yee

References

Kleiber, C. and Kotz, S. (2003). *Statistical Size Distributions in Economics and Actuarial Sciences*, Hoboken, NJ, USA: Wiley-Interscience.

See Also

Sinmad, genbetaII, betaII, dagum, fisk, inv.lomax, lomax, paralogistic, inv.paralogistic, simulate.vlm.

Examples

```
sdata <- data.frame(y = rsinmad(n = 1000, shape1 = exp(1),
                    scale = exp(2), shape3 = exp(0)))
fit <- vglm(y ~ 1, sinmad(lss = FALSE), sdata, trace = TRUE)
fit <- vglm(y ~ 1, sinmad(lss = FALSE, ishape1.a = exp(1)),
            sdata, trace = TRUE)
coef(fit, matrix = TRUE)
Coef(fit)
summary(fit)

# Harder problem (has the shape3.q parameter going to infinity)

set.seed(3)
sdata <- data.frame(y1 = rbeta(1000, 6, 6))
# hist(with(sdata, y1))
if (FALSE) {
# These struggle
  fit1 <- vglm(y1 ~ 1, sinmad(lss = FALSE), sdata, trace = TRUE)
  fit1 <- vglm(y1 ~ 1, sinmad(lss = FALSE), sdata, trace = TRUE,
               crit = "coef")
  Coef(fit1)
}
# Try this remedy:
fit2 <- vglm(y1 ~ 1, data = sdata, trace = TRUE, stepsize = 0.05, maxit = 99,
             sinmad(lss = FALSE, ishape3.q = 3, lshape3.q = "logloglink"))

coef(fit2, matrix = TRUE)
Coef(fit2)
```

Skellam *The Skellam Distribution*

Description

Density and random generation for the Skellam distribution.

Usage

```
dskellam(x, mu1, mu2, log = FALSE)
rskellam(n, mu1, mu2)
```

Arguments

x	vector of quantiles.
n	number of observations. Same as runif.
mu1, mu2	See skellam
.	
log	Logical; if TRUE, the logarithm is returned.

Details

See skellam, the **VGAM** family function for estimating the parameters, for the formula of the probability density function and other details.

Value

`dskellam` gives the density, and `rskellam` generates random deviates.

Warning

Numerical problems may occur for data if μ_1 and/or μ_2 are large. The normal approximation for this case has not been implemented yet.

See Also

skellam, dpois.

Examples

```
## Not run:  mu1 <- 1; mu2 <- 2; x <- (-7):7
plot(x, dskellam(x, mu1, mu2), type = "h", las = 1, col = "blue",
     main = paste("Density of Skellam distribution with mu1 = ", mu1,
                  " and mu2 = ", mu2, sep = ""))
## End(Not run)
```

---

skellam *Skellam Distribution Family Function*

Description

Estimates the two parameters of a Skellam distribution by maximum likelihood estimation.

Usage

```
skellam(lmu1 = "loglink", lmu2 = "loglink", imu1 = NULL, imu2 = NULL,
        nsimEIM = 100, parallel = FALSE, zero = NULL)
```

Arguments

`lmu1, lmu2` Link functions for the μ_1 and μ_2 parameters. See `Links` for more choices and for general information.

`imu1, imu2` Optional initial values for the parameters. See `CommonVGAMffArguments` for more information. If convergence failure occurs (this **VGAM** family function seems to require good initial values) try using these arguments.

`nsimEIM, parallel, zero`
See `CommonVGAMffArguments` for information. In particular, setting `parallel=TRUE` will constrain the two means to be equal.

Details

The Skellam distribution models the difference between two independent Poisson distributions (with means μ_j, say). It has density function

$$f(y; \mu_1, \mu_2) = \left(\frac{\mu_1}{\mu_2}\right)^{y/2} \exp(-\mu_1 - \mu_2)\, I_{|y|}(2\sqrt{\mu_1 \mu_2})$$

where y is an integer, $\mu_1 > 0$, $\mu_2 > 0$. Here, I_v is the modified Bessel function of the first kind with order v.

The mean is $\mu_1 - \mu_2$ (returned as the fitted values), and the variance is $\mu_1 + \mu_2$. Simulated Fisher scoring is implemented.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Warning

This **VGAM** family function seems fragile and very sensitive to the initial values. Use very cautiously!!

Note

Numerical problems may occur for data if μ_1 and/or μ_2 are large.

References

Skellam, J. G. (1946). The frequency distribution of the difference between two Poisson variates belonging to different populations. *Journal of the Royal Statistical Society, Series A*, **109**, 296.

See Also

`dskellam`, `dpois`, `poissonff`.

Examples

```
## Not run:
sdata <- data.frame(x2 = runif(nn <- 1000))
sdata <- transform(sdata, mu1 = exp(1 + x2), mu2 = exp(1 + x2))
sdata <- transform(sdata, y = rskellam(nn, mu1, mu2))
fit1 <- vglm(y ~ x2, skellam, data = sdata, trace = TRUE, crit = "coef")
fit2 <- vglm(y ~ x2, skellam(parallel = TRUE), data = sdata, trace = TRUE)
coef(fit1, matrix = TRUE)
coef(fit2, matrix = TRUE)
summary(fit1)
# Likelihood ratio test for equal means:
pchisq(2 * (logLik(fit1) - logLik(fit2)),
       df = df.residual(fit2) - df.residual(fit1), lower.tail = FALSE)
lrtest(fit1, fit2)  # Alternative

## End(Not run)
```

---

skewnorm *Skew-Normal Distribution*

Description

Density and random generation for the univariate skew-normal distribution.

Usage

```
dskewnorm(x, location = 0, scale = 1, shape = 0, log = FALSE)
rskewnorm(n, location = 0, scale = 1, shape = 0)
```

Arguments

`x`	vector of quantiles.
`n`	number of observations. Same as `runif`.
`location`	The location parameter ξ. A vector.
`scale`	The scale parameter ω. A positive vector.
`shape`	The shape parameter. It is called α in `skewnormal`.
`log`	Logical. If `log=TRUE` then the logarithm of the density is returned.

Details

See `skewnormal`, which currently only estimates the shape parameter. More generally here, $Z = \xi + \omega Y$ where Y has a standard skew-normal distribution (see `skewnormal`), ξ is the location parameter and ω is the scale parameter.

Value

`dskewnorm` gives the density, `rskewnorm` generates random deviates.

Note

The default values of all three parameters corresponds to the skew-normal being the standard normal distribution.

Author(s)

T. W. Yee

References

http://tango.stat.unipd.it/SN.

See Also

skewnormal.

Examples

```
## Not run:  N <- 200  # Grid resolution
shape <- 7; x <- seq(-4, 4, len = N)
plot(x, dskewnorm(x, shape = shape), type = "l", col = "blue", las = 1,
     ylab = "", lty = 1, lwd = 2)
abline(v = 0, h = 0, col = "grey")
lines(x, dnorm(x), col = "orange", lty = 2, lwd = 2)
legend("topleft", leg = c(paste("Blue = dskewnorm(x, ", shape,")", sep = ""),
       "Orange = standard normal density"), lty = 1:2, lwd = 2,
       col = c("blue", "orange"))
## End(Not run)
```

---

skewnormal *Univariate Skew-Normal Distribution Family Function*

Description

Maximum likelihood estimation of the shape parameter of a univariate skew-normal distribution.

Usage

```
skewnormal(lshape = "identitylink", ishape = NULL, nsimEIM = NULL)
```

Arguments

lshape, ishape, nsimEIM
: See Links and CommonVGAMffArguments.

Details

The univariate skew-normal distribution has a density function that can be written

$$f(y) = 2\,\phi(y)\,\Phi(\alpha y)$$

where α is the shape parameter. Here, ϕ is the standard normal density and Φ its cumulative distribution function. When $\alpha = 0$ the result is a standard normal distribution. When $\alpha = 1$ it models the distribution of the maximum of two independent standard normal variates. When the absolute value of the shape parameter increases the skewness of the distribution increases. The limit as the shape parameter tends to positive infinity results in the folded normal distribution or half-normal distribution. When the shape parameter changes its sign, the density is reflected about $y = 0$.

The mean of the distribution is $\mu = \alpha\sqrt{2/(\pi(1+\alpha^2))}$ and these are returned as the fitted values. The variance of the distribution is $1 - \mu^2$. The Newton-Raphson algorithm is used unless the `nsimEIM` argument is used.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Warning

It is well known that the EIM of Azzalini's skew-normal distribution is singular for skewness parameter tending to zero, and thus produces influential problems.

Note

It is a good idea to use several different initial values to ensure that the global solution is obtained.

This family function will be modified (hopefully soon) to handle a location and scale parameter too.

Author(s)

Thomas W. Yee

References

Azzalini, A. A. (1985). A class of distributions which include the normal. *Scandinavian Journal of Statistics*, **12**, 171–178.

Azzalini, A. and Capitanio, A. (1999). Statistical applications of the multivariate skew-normal distribution. *Journal of the Royal Statistical Society, Series B, Methodological*, **61**, 579–602.

See Also

`skewnorm`, `uninormal`, `foldnormal`.

Examples

```
sdata <- data.frame(y1 = rskewnorm(nn <- 1000, shape = 5))
fit1 <- vglm(y1 ~ 1, skewnormal, data = sdata, trace = TRUE)
coef(fit1, matrix = TRUE)
head(fitted(fit1), 1)
with(sdata, mean(y1))
## Not run:  with(sdata, hist(y1, prob = TRUE))
x <- with(sdata, seq(min(y1), max(y1), len = 200))
with(sdata, lines(x, dskewnorm(x, shape = Coef(fit1)), col = "blue"))
## End(Not run)

sdata <- data.frame(x2 = runif(nn))
sdata <- transform(sdata, y2 = rskewnorm(nn, shape = 1 + 2*x2))
fit2 <- vglm(y2 ~ x2, skewnormal, data = sdata, trace = TRUE, crit = "coef")
summary(fit2)
```

`Slash` *Slash Distribution*

Description

Density function, distribution function, and random generation for the slash distribution.

Usage

```
dslash(x, mu = 0, sigma = 1, log = FALSE,
       smallno = .Machine$double.eps*1000)
pslash(q, mu = 0, sigma = 1, very.negative = -10000,
       lower.tail = TRUE, log.p = FALSE)
rslash(n, mu = 0, sigma = 1)
```

Arguments

`x, q`	vector of quantiles.
`n`	Same as `runif`.
`mu, sigma`	the mean and standard deviation of the univariate normal distribution.
`log`	Logical. If `TRUE` then the logarithm of the density is returned.
`very.negative`	Numeric, of length 1. A large negative value. For `(q-mu)/sigma` values less than this, the value 0 is returned because `integrate` tends to fail. A warning is issued. Similarly, if `(q-mu)/sigma` is greater than `abs(very.negative)` then 1 is returned with a warning.
`smallno`	See `slash`.
`lower.tail, log.p`	Same meaning as in `pnorm` or `qnorm`.

Details

See `slash`, the **VGAM** family function for estimating the two parameters by maximum likelihood estimation, for the formula of the probability density function and other details.

Function `pslash` uses a `for ()` loop and `integrate`, meaning it's very slow. It may also be inaccurate for extreme values of q, and returns with 1 or 0 values when too extreme compared to `very.negative`.

Value

`dslash` gives the density, and `pslash` gives the distribution function, `rslash` generates random deviates.

Note

`pslash` is very slow.

Author(s)

Thomas W. Yee and C. S. Chee

See Also

`slash`.

Examples

```
## Not run:
curve(dslash, col = "blue", ylab = "f(x)", -5, 5, ylim = c(0, 0.4), las = 1,
     main = "Standard slash, normal and Cauchy densities", lwd = 2)
curve(dnorm, col = "black", lty = 2, lwd = 2, add = TRUE)
curve(dcauchy, col = "orange", lty = 3, lwd = 2, add = TRUE)
legend("topleft", c("slash", "normal", "Cauchy"), lty = 1:3,
       col = c("blue","black","orange"), lwd = 2)

curve(pslash, col = "blue", -5, 5, ylim = 0:1)
pslash(c(-Inf, -20000, 20000, Inf))  # Gives a warning

## End(Not run)
```

`slash` *Slash Distribution Family Function*

Description

Estimates the two parameters of the slash distribution by maximum likelihood estimation.

Usage

```
slash(lmu = "identitylink", lsigma = "loglink",
      imu = NULL, isigma = NULL, gprobs.y = ppoints(8), nsimEIM = 250,
      zero = NULL, smallno = .Machine$double.eps*1000)
```

Arguments

lmu, lsigma	Parameter link functions applied to the μ and σ parameters, respectively. See Links for more choices.
imu, isigma	Initial values. A NULL means an initial value is chosen internally. See CommonVGAMffArguments for more information.
gprobs.y	Used to compute the initial values for mu. This argument is fed into the probs argument of quantile to construct a grid, which is used to evaluate the log-likelihood. This must have values between 0 and 1.
nsimEIM, zero	See CommonVGAMffArguments for information.
smallno	Small positive number, used to test for the singularity.

Details

The standard slash distribution is the distribution of the ratio of a standard normal variable to an independent standard uniform(0,1) variable. It is mainly of use in simulation studies. One of its properties is that it has heavy tails, similar to those of the Cauchy.

The general slash distribution can be obtained by replacing the univariate normal variable by a general normal $N(\mu, \sigma)$ random variable. It has a density that can be written as

$$f(y) = \left\{ \begin{array}{cl} 1/(2\sigma\sqrt(2\pi)) & if y = \mu, \\ 1 - \exp(-(((y-\mu)/\sigma)^2)/2))/(\sqrt(2pi)\sigma((y-\mu)/\sigma)^2) & if y \neq \mu. \end{array} \right.$$

where μ and σ are the mean and standard deviation of the univariate normal distribution respectively.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Note

Fisher scoring using simulation is used. Convergence is often quite slow. Numerical problems may occur.

Author(s)

T. W. Yee and C. S. Chee

References

Johnson, N. L. and Kotz, S. and Balakrishnan, N. (1994). *Continuous Univariate Distributions*, 2nd edition, Volume 1, New York: Wiley.

Kafadar, K. (1982). A Biweight Approach to the One-Sample Problem *Journal of the American Statistical Association*, **77**, 416–424.

See Also

`rslash`, `simulate.vlm`.

Examples

```
## Not run:
sdata <- data.frame(y = rslash(n = 1000, mu = 4, sigma = exp(2)))
fit <- vglm(y ~ 1, slash, data = sdata, trace = TRUE)
coef(fit, matrix = TRUE)
Coef(fit)
summary(fit)

## End(Not run)
```

---

sloglink — *Square root–Log Link Mixtures*

Description

Computes some square root–log mixture link transformations, including their inverse and the first few derivatives.

Usage

```
  sloglink(theta, bvalue = NULL, taumix.log = 1, tol = 1e-13,
    nmax = 99, inverse = FALSE, deriv = 0, short = TRUE,
    tag = FALSE, c10 = c(2, -2))
lcsloglink(theta, bvalue = NULL, pmix.log = 0.01, tol = 1e-13,
    nmax = 99, inverse = FALSE, deriv = 0, short = TRUE,
    tag = FALSE, c10 = c(2, -2))
```

Arguments

`theta`	Numeric or character. See below for further details.
`bvalue`	See `Links`.
`taumix.log`	Numeric, of length 1. Mixing parameter directed at `loglink`. Then `1 - exp(-taumix.log * theta)` is used to weight `sqrtlink`. Thus a 0 value will result in `loglink`, and a very large numeric such as `1e4` should be roughly equivalent to `sqrtlink` over almost all of the parameter space.

pmix.log — Numeric, of length 1. Mixing probability assigned to loglink. Then 1 - pmix.log is used to weight sqrtlink. Thus a 0 value will result in sqrtlink and 1 is equivalent to loglink.

tol, nmax — Arguments fed into a function implementing a vectorized bisection method.

inverse, deriv, short, tag — Details at Links.

c10 — See sqrtlink and loglink.

Details

For general information see alogitlink.

Value

The following holds for the *linear combination* (LC) variant. For deriv = 0, (1 - pmix.log) * sqrtlink(mu, c10 = c10) + pmix.log * loglink(mu) when inverse = FALSE, and if inverse = TRUE then a nonlinear equation is solved for mu, given eta passed in as theta. For deriv = 1, then the function returns *d* eta / *d* theta as a function of theta if inverse = FALSE, else if inverse = TRUE then it returns the reciprocal.

Warning

The default values for taumix.log and pmix.log may change in the future. The name and order of the arguments may change too.

Author(s)

Thomas W. Yee

See Also

alogitlink, sqrtlink, loglink, Links, poissonff, hdeff.

Examples

```
mu  <- seq(0.01, 3, length = 10)
sloglink(mu)
max(abs(sloglink(sloglink(mu), inv = TRUE) - mu))  # 0?
```

---

sm.os — *Defining O'Sullivan Spline Smooths in VGAM Formulas*

Description

This function represents an O-spline smooth term in a vgam formula and confers automatic smoothing parameter selection.

Usage

```
sm.os(x, ..., niknots = 6, spar = -1, o.order = 2,
      alg.niknots = c("s", ".nknots.smspl")[1], all.knots = FALSE,
      ridge.adj = 1e-5, spillover = 0.01, maxspar = 1e12,
      outer.ok = FALSE, fixspar = FALSE)
```

Arguments

`x`	covariate (abscissae) to be smoothed. Also called the regressor. If the `xij` facility is used then these covariates are inputted via the `...` argument.
`...`	Used to accommodate the other $M - 1$ covariates when the `xij` facility is used. See Section 3.4.4 of Yee (2015) for something very similar. This argument, found in the second argument, means that the other argument names must be fully specified if used, e.g., `outer.ok` and not `outer`. See the example below. In the example below, the term in the main formula is `sm.os(gcost.air, gcost.trn, gcost.bus)` and one might be tempted to use something like `sm.os(gcost)` to represent that `xij` term. However, this is not recommended because `sm.os(gcost)` might not have the same number of columns as `sm.os(gcost.air, gcost.trn, gcost.bus)` etc. That is, it is best to select one of the diagonal elements of the block matrix to represent that term.
`niknots`	numeric, the number of *interior* knots, called K below. The default is to use this value. If you want `alg.niknots` to operate then assign `NULL` to this argument.
`alg.niknots`	character. The algorithm used to determine the number of interior knots. Only used when `all.knots = FALSE` and `niknots = NULL`. Note that `".nknots.smspl"` corresponds to the default of `smooth.spline`. The value `"s"` corresponds to the same algorithm as `s`.
`all.knots`	logical. If `TRUE` then all distinct points in `x` are used as the interior knots. If `FALSE` (default) then a subset of `x[]` is used, specifically `x[j]` where the `niknots` indices are quantiles that are evenly spaced with respect to the argument `probs`—see `quantile`. If `all.knots = FALSE` and `niknots = NULL` then the argument `alg.niknots` is used to compute `niknots`.
`spar, maxspar`	`spar` is a vector of smoothing parameters. Negative values mean that `magic` will choose initial values in order to do the optimization at each P-IRLS iteration. Positive values mean that they are used as initial values for `magic`. If `fixspar = TRUE` then `spar` should be assigned a vector of positive values (but having values less than `maxspar`); then the smoothing parameters will be fixed and `magic` will not be used.
`o.order`	The order of the O'Sullivan penalzed spline. Any one value from `1:4` is acceptable. The degree of the spline is `2 * o.order - 1`, so that cubic splines are the default. Setting `o.order = 1` results in a linear spline which is a piecewise linear function.
`ridge.adj`	small positive number to stabilize linear dependencies among B-spline bases.
`spillover`	small and positive proportion of the range used on the outside of the boundary values. This defines the endpoints a and b that cover the data x_i, i.e., we are interested in the interval $[a, b]$ which contains all the abscissae. The interior knots are strictly inside (a, b).

outer.ok Fed into the argument (by the same name) of splineDesign.

fixspar logical. If TRUE then spar should be a vector with positive values and the smoothing parameters are fixed at those values. If FALSE then spar contains the initial values for the smoothing parameters, and magic is called to determine (hopefully) some good values for the smoothing parameters.

Details

This function is currently used by vgam to allow automatic smoothing parameter selection based on O-splines to minimize an UBRE quantity. In contrast, s operates by having a prespecified amount of smoothing, e.g., its df argument. When the sample size is reasonably large this function is recommended over s also because backfitting is not required. This function therefore allows 2nd-generation VGAMs to be fitted (called G2-VGAMs, or Penalized-VGAMs).

This function should only be used with vgam. This function uses quantile to choose the knots, whereas sm.ps chooses equally-spaced knots. As Wand and Ormerod (2008) write, in most situations the differences will be minor, but it is possible for problems to arise for either strategy by constructing certain regression functions and predictor variable distributions. Any differences between O-splines and P-splines tend to be at the boundaries. O-splines have *natural boundary constraints* so that the solution is linear beyond the boundary knots.

Some arguments in decreasing order of precedence are: all.knots, niknots, alg.niknots.

Unlike s, which is symbolic and does not perform any smoothing itself, this function does compute the penalized spline when used by vgam—it creates the appropriate columns of the model matrix. When this function is used within vgam, automatic smoothing parameter selection is implemented by calling magic after the necessary link-ups are done.

By default this function centres the component function. This function is also *smart*; it can be used for smart prediction (Section 18.6 of Yee (2015)). Automatic smoothing parameter selection is performed using *performance-oriented iteration* whereby an optimization problem is solved at each IRLS iteration.

This function works better when the sample size is large, e.g., when in the hundreds, say.

Value

A matrix with attributes that are (only) used by vgam. The number of rows of the matrix is length(x). The number of columns is a function of the number of interior knots K and the order of the O-spline m: $K + 2m - 1$. In code, this is niknots + 2 * o.order - 1, or using sm.ps-like arguments, ps.int + degree - 1 (where ps.int should be more generally interpreted as the number of intervals. The formula is the same as sm.ps.). It transpires then that sm.os and sm.ps are very similar.

Warning

Being introduced into **VGAM** for the first time, this function (and those associated with it) should be used cautiously. Not all options are fully working or have been tested yet, and there are bound to be some bugs lurking around.

Note

This function is currently under development and may change in the future.

One might try using this function with vglm so as to fit a regression spline, however, the default value of niknots will probably be too high for most data sets.

Author(s)

T. W. Yee, with some of the essential R code coming from the appendix of Wand and Ormerod (2008).

References

Wand, M. P. and Ormerod, J. T. (2008). On semiparametric regression with O'Sullivan penalized splines. *Australian and New Zealand Journal of Statistics*, **50**(2): 179–198.

See Also

vgam, sm.ps, s, smartpred, is.smart, summarypvgam, smooth.spline, splineDesign, bs, magic.

Examples

```
sm.os(runif(20))

## Not run:
data("TravelMode", package = "AER")  # Need to install "AER" first
air.df <- subset(TravelMode, mode == "air")  # Form 4 smaller data frames
bus.df <- subset(TravelMode, mode == "bus")
trn.df <- subset(TravelMode, mode == "train")
car.df <- subset(TravelMode, mode == "car")
TravelMode2 <- data.frame(income     = air.df$income,
                          wait.air   = air.df$wait  - car.df$wait,
                          wait.trn   = trn.df$wait  - car.df$wait,
                          wait.bus   = bus.df$wait  - car.df$wait,
                          gcost.air  = air.df$gcost - car.df$gcost,
                          gcost.trn  = trn.df$gcost - car.df$gcost,
                          gcost.bus  = bus.df$gcost - car.df$gcost,
                          wait       = air.df$wait)  # Value is unimportant
TravelMode2$mode <- subset(TravelMode, choice == "yes")$mode  # The response
TravelMode2 <- transform(TravelMode2, incom.air = income, incom.trn = 0,
                                      incom.bus = 0)
set.seed(1)
TravelMode2 <- transform(TravelMode2,
                         junkx2 = runif(nrow(TravelMode2)))

tfit2 <-
  vgam(mode ~ sm.os(gcost.air, gcost.trn, gcost.bus) + ns(junkx2, 4) +
              sm.os(incom.air, incom.trn, incom.bus) + wait ,
       crit = "coef",
       multinomial(parallel = FALSE ~ 1), data = TravelMode2,
       xij = list(sm.os(gcost.air, gcost.trn, gcost.bus) ~
                  sm.os(gcost.air, gcost.trn, gcost.bus) +
```

```
                  sm.os(gcost.trn, gcost.bus, gcost.air) +
                  sm.os(gcost.bus, gcost.air, gcost.trn),
                  sm.os(incom.air, incom.trn, incom.bus) ~
                  sm.os(incom.air, incom.trn, incom.bus) +
                  sm.os(incom.trn, incom.bus, incom.air) +
                  sm.os(incom.bus, incom.air, incom.trn),
                  wait   ~  wait.air +  wait.trn +  wait.bus),
       form2 = ~  sm.os(gcost.air, gcost.trn, gcost.bus) +
                  sm.os(gcost.trn, gcost.bus, gcost.air) +
                  sm.os(gcost.bus, gcost.air, gcost.trn) +
                  wait +
                  sm.os(incom.air, incom.trn, incom.bus) +
                  sm.os(incom.trn, incom.bus, incom.air) +
                  sm.os(incom.bus, incom.air, incom.trn) +
                  junkx2 + ns(junkx2, 4) +
                  incom.air + incom.trn + incom.bus +
                  gcost.air + gcost.trn + gcost.bus +
                  wait.air +  wait.trn +  wait.bus)
par(mfrow = c(2, 2))
plot(tfit2, se = TRUE, lcol = "orange", scol = "blue", ylim = c(-4, 4))
summary(tfit2)

## End(Not run)
```

---

sm.ps *Defining Penalized Spline Smooths in VGAM Formulas*

Description

This function represents a P-spline smooth term in a vgam formula and confers automatic smoothing parameter selection.

Usage

```
sm.ps(x, ..., ps.int = NULL, spar = -1, degree = 3, p.order = 2,
      ridge.adj = 1e-5, spillover = 0.01, maxspar = 1e12,
      outer.ok = FALSE, mux = NULL, fixspar = FALSE)
```

Arguments

x, ... See sm.os.

ps.int the number of equally-spaced B-spline intervals. Note that the number of knots is equal to ps.int + 2*degree + 1. The default, signified by NULL, means that the maximum of the value 7 and degree is chosen. This usually means 6 interior knots for big data sets. However, if this is too high compared to the length of x, then some adjustment is made. In the case where mux is assigned a numerical value (suggestions: some value between 1 and 2) then ceiling(mux * log(length(unique(x.index)))) is used, where x.index is the combined data. No matter what, the above is not guaranteed to work on every data set. This argument may change in the future. See also argument mux.

spar, maxspar — See sm.os.

mux — numeric. If given, then this argument multiplies log(length(unique(x))) to obtain ps.int. If ps.int is given then this argument is ignored.

degree — degree of B-spline basis. Usually this will be 2 or 3; and the values 1 or 4 might possibly be used.

p.order — order of difference penalty (0 is the ridge penalty).

ridge.adj, spillover — See sm.os.

outer.ok, fixspar — See sm.os.

Details

This function can be used by vgam to allow automatic smoothing parameter selection based on P-splines and minimizing an UBRE quantity.

This function should only be used with vgam and is an alternative to sm.os; see that function for some details that also apply here.

Value

A matrix with attributes that are (only) used by vgam. The number of rows of the matrix is length(x) and the number of columns is ps.int + degree - 1. The latter is because the function is centred.

Warning

See sm.os.

Note

This function is currently under development and may change in the future. In particular, the default for ps.int is subject to change.

Author(s)

B. D. Marx wrote the original function. Subsequent edits were made by T. W. Yee and C. Somchit.

References

Eilers, P. H. C. and Marx, B. D. (1996). Flexible smoothing with B-splines and penalties (with comments and rejoinder). *Statistical Science*, **11**(2): 89–121.

See Also

sm.os, s, vgam, smartpred, is.smart, summarypvgam, splineDesign, bs, magic.

Examples

```
sm.ps(runif(20))
sm.ps(runif(20), ps.int = 5)

## Not run:
data("TravelMode", package = "AER")  # Need to install "AER" first
air.df <- subset(TravelMode, mode == "air")  # Form 4 smaller data frames
bus.df <- subset(TravelMode, mode == "bus")
trn.df <- subset(TravelMode, mode == "train")
car.df <- subset(TravelMode, mode == "car")
TravelMode2 <- data.frame(income     = air.df$income,
                          wait.air   = air.df$wait  - car.df$wait,
                          wait.trn   = trn.df$wait  - car.df$wait,
                          wait.bus   = bus.df$wait  - car.df$wait,
                          gcost.air  = air.df$gcost - car.df$gcost,
                          gcost.trn  = trn.df$gcost - car.df$gcost,
                          gcost.bus  = bus.df$gcost - car.df$gcost,
                          wait       = air.df$wait)  # Value is unimportant
TravelMode2$mode <- subset(TravelMode, choice == "yes")$mode  # The response
TravelMode2 <- transform(TravelMode2, incom.air = income, incom.trn = 0,
                                      incom.bus = 0)
set.seed(1)
TravelMode2 <- transform(TravelMode2,
                         junkx2 = runif(nrow(TravelMode2)))

tfit2 <-
  vgam(mode ~ sm.ps(gcost.air, gcost.trn, gcost.bus) + ns(junkx2, 4) +
              sm.ps(incom.air, incom.trn, incom.bus) + wait ,
       crit = "coef",
       multinomial(parallel = FALSE ~ 1), data = TravelMode2,
       xij = list(sm.ps(gcost.air, gcost.trn, gcost.bus) ~
                  sm.ps(gcost.air, gcost.trn, gcost.bus) +
                  sm.ps(gcost.trn, gcost.bus, gcost.air) +
                  sm.ps(gcost.bus, gcost.air, gcost.trn),
                  sm.ps(incom.air, incom.trn, incom.bus) ~
                  sm.ps(incom.air, incom.trn, incom.bus) +
                  sm.ps(incom.trn, incom.bus, incom.air) +
                  sm.ps(incom.bus, incom.air, incom.trn),
                  wait   ~  wait.air +  wait.trn +  wait.bus),
       form2 = ~  sm.ps(gcost.air, gcost.trn, gcost.bus) +
                  sm.ps(gcost.trn, gcost.bus, gcost.air) +
                  sm.ps(gcost.bus, gcost.air, gcost.trn) +
                  wait +
                  sm.ps(incom.air, incom.trn, incom.bus) +
                  sm.ps(incom.trn, incom.bus, incom.air) +
                  sm.ps(incom.bus, incom.air, incom.trn) +
                  junkx2 + ns(junkx2, 4) +
                  incom.air + incom.trn + incom.bus +
                  gcost.air + gcost.trn + gcost.bus +
                  wait.air +  wait.trn +  wait.bus)
par(mfrow = c(2, 2))
plot(tfit2, se = TRUE, lcol = "orange", scol = "blue", ylim = c(-4, 4))
```

```
summary(tfit2)

## End(Not run)
```

---

smart.expression *S Expression for Smart Functions*

Description

`smart.expression` is an S expression for a smart function to call itself. It is best if you go through it line by line, but most users will not need to know anything about it. It requires the primary argument of the smart function to be called `"x"`.

The list component `match.call` must be assigned the value of `match.call()` in the smart function; this is so that the smart function can call itself later.

See Also

`match.call`.

Examples

```
print(sm.min2)
```

---

smart.mode.is *Determine What Mode the Smart Prediction is In*

Description

Determine which of three modes the smart prediction is currently in.

Usage

```
smart.mode.is(mode.arg = NULL)
```

Arguments

`mode.arg`	a character string, either `"read"`, `"write"` or `"neutral"`.

Details

Smart functions such as `bs` and `poly` need to know what mode smart prediction is in. If it is in `"write"` mode then the parameters are saved to `.smart.prediction` using `put.smart`. If in `"read"` mode then the parameters are read in using `get.smart`. If in `"neutral"` mode then the smart function behaves like an ordinary function.

Value

If mode.arg is given, then either TRUE or FALSE is returned. If mode.arg is not given, then the mode ("neutral", "read" or "write") is returned. Usually, the mode is "neutral".

See Also

put.smart, bs, poly.

Examples

```
print(sm.min1)
smart.mode.is()  # Returns "neutral"
smart.mode.is(smart.mode.is())  # Returns TRUE
```

---

smartpred *Smart Prediction*

Description

Data-dependent parameters in formula terms can cause problems in when predicting. The **smartpred** package saves data-dependent parameters on the object so that the bug is fixed. The lm and glm functions have been fixed properly. Note that the **VGAM** package by T. W. Yee automatically comes with smart prediction.

Usage

```
sm.bs(x, df = NULL, knots = NULL, degree = 3, intercept = FALSE,
      Boundary.knots = range(x))
sm.ns(x, df = NULL, knots = NULL, intercept = FALSE,
      Boundary.knots = range(x))
sm.poly(x, ..., degree = 1, coefs = NULL, raw = FALSE)
sm.scale(x, center = TRUE, scale = TRUE)
```

Arguments

x	The x argument is actually common to them all.

df, knots, intercept, Boundary.knots
	See bs and/or ns.

degree, ..., coefs, raw
	See poly.

center, scale	See scale.

Details

R version 1.6.0 introduced a partial fix for the prediction problem because it does not work all the time, e.g., for terms such as `I(poly(x, 3))`, `poly(c(scale(x)), 3)`, `bs(scale(x), 3)`, `scale(scale(x))`. See the examples below. Smart prediction, however, will always work.

The basic idea is that the functions in the formula are now smart, and the modelling functions make use of these smart functions. Smart prediction works in two ways: using `smart.expression`, or using a combination of `put.smart` and `get.smart`.

Value

The usual value returned by `bs`, `ns`, `poly` and `scale`, When used with functions such as `vglm` the data-dependent parameters are saved on one slot component called `smart.prediction`.

Side Effects

The variables `.max.smart`, `.smart.prediction` and `.smart.prediction.counter` are created while the model is being fitted. They are created in a new environment called `smartpredenv`. These variables are deleted after the model has been fitted. However, if there is an error in the model fitting function or the fitting model is killed (e.g., by typing control-C) then these variables will be left in `smartpredenv`. At the beginning of model fitting, these variables are deleted if present in `smartpredenv`.

During prediction, the variables `.smart.prediction` and `.smart.prediction.counter` are reconstructed and read by the smart functions when the model frame is re-evaluated. After prediction, these variables are deleted.

If the modelling function is used with argument `smart = FALSE` (e.g., `vglm(..., smart = FALSE)`) then smart prediction will not be used, and the results should match with the original R functions.

WARNING

The functions `bs`, `ns`, `poly` and `scale` are now left alone (from 2014-05 onwards) and no longer smart. They work via safe prediction. The smart versions of these functions have been renamed and they begin with `"sm."`.

The functions `predict.bs` and `predict.ns` are not smart. That is because they operate on objects that contain attributes only and do not have list components or slots. The function `predict.poly` is not smart.

Author(s)

T. W. Yee and T. J. Hastie

See Also

`get.smart.prediction`, `get.smart`, `put.smart`, `smart.expression`, `smart.mode.is`, `setup.smart`, `wrapup.smart`. For `vgam` in **VGAM**, `sm.ps` is important. Commonly used data-dependent functions include `scale`, `poly`, `bs`, `ns`. In R, the functions `bs` and `ns` are in the **splines** package, and this library is automatically loaded in because it contains compiled code that `bs` and `ns` call.

The functions `vglm`, `vgam`, `rrvglm` and `cqo` in T. W. Yee's **VGAM** package are examples of modelling functions that employ smart prediction.

Examples

```
# Create some data first
n <- 20
set.seed(86)  # For reproducibility of the random numbers
ldata <- data.frame(x2 = sort(runif(n)), y = sort(runif(n)))
library("splines")  # To get ns() in R

# This will work for R 1.6.0 and later
fit <- lm(y ~ ns(x2, df = 5), data = ldata)
## Not run:
plot(y ~ x2, data = ldata)
lines(fitted(fit) ~ x2, data = ldata)
new.ldata <- data.frame(x2 = seq(0, 1, len = n))
points(predict(fit, new.ldata) ~ x2, new.ldata, type = "b", col = 2, err = -1)

## End(Not run)

# The following fails for R 1.6.x and later. It can be
# made to work with smart prediction provided
# ns is changed to sm.ns and scale is changed to sm.scale:
fit1 <- lm(y ~ ns(scale(x2), df = 5), data = ldata)
## Not run:
plot(y ~ x2, data = ldata, main = "Safe prediction fails")
lines(fitted(fit1) ~ x2, data = ldata)
points(predict(fit1, new.ldata) ~ x2, new.ldata, type = "b", col = 2, err = -1)

## End(Not run)

# Fit the above using smart prediction
## Not run:
library("VGAM")  # The following requires the VGAM package to be loaded
fit2 <- vglm(y ~ sm.ns(sm.scale(x2), df = 5), uninormal, data = ldata)
fit2@smart.prediction
plot(y ~ x2, data = ldata, main = "Smart prediction")
lines(fitted(fit2) ~ x2, data = ldata)
points(predict(fit2, new.ldata, type = "response") ~ x2, data = new.ldata,
       type = "b", col = 2, err = -1)

## End(Not run)
```

`specials` *Special Values or Quantities in a Fitted Object*

Description

Return any special values or quantities in a fitted object, and in particular in a VGLM fit

Usage

```
specials(object, ...)
specialsvglm(object, ...)
```

Arguments

`object`	an object of class `"vglm"` whose family function begins with `"gait"`.
`...`	any additional arguments, to future-proof this function.

Details

This extractor function was motivated by GAITD regression (Yee and Ma, 2024) where the values from three disjoint sets are referred to as *special*. More generally, S4 methods functions can be written so that `specials()` will work on any S4 object, where what is called special depends on the methodology at hand.

Value

Returns any 'special' values or quantities associated with a fitted regression model. This is often something simple such as a list or a vector.

References

Yee, T. W. and Ma, C. (2024). Generally altered, inflated, truncated and deflated regression. *Statistical Science*, **39** (in press).

See Also

`vglm`, `vglm-class`, `inflated`, `altered`, `truncated`, `Gaitdpois`, `gaitdpoisson`.

Examples

```
abdata <- data.frame(y = 0:7, w = c(182, 41, 12, 2, 2, 0, 0, 1))
fit1 <- vglm(y ~ 1, gaitdpoisson(a.mix = 0), data = abdata,
             weight = w, subset = w > 0)
specials(fit1)
```

`spikeplot` *Spike Plot*

Description

Produces a spike plot of a numeric vector.

Usage

```
spikeplot(x, freq = FALSE, as.table = FALSE, col = par("col"),
    lty = par("lty"), lwd = par("lwd"), lend = par("lend"),
    type = "h", xlab = deparse1(substitute(x)), ylab = NULL,
    capped = FALSE, cex = sqrt(lwd) / 2, pch = 19, pcol = col, scol = NULL,
    slty = NULL, slwd = NULL, new.plot = TRUE, offset.x = 0, ymux = 1, ...)
```

Arguments

- `x` Numeric, passed into `table`.
- `freq` Logical. If `TRUE` then the y-axis measures the frequencies, else the sample proportions. Intended to be as `hist`.
- `as.table` Logical. If `TRUE` then the call to `plot` is closer to `plot(table(x), ...)`, meaning the labelling differs from `as.table = FALSE`. The default is to convert `table(x)` into a numeric vector which is then passed into `plot` so that the labelling is more uniform along the x-axis.
- `col, type, lty, lwd` See `par`.
- `lend, xlab, ylab` See `par`.
- `capped, cex, pch, pcol` First argument is logical. If `TRUE` then the others argument are used to place points at the top using `points` with `pcol` being its colour. See `par`.
- `scol, slty, slwd` Similar to `col`, `lty` and `lwd` but apply to some selected values. The input may be a named list such as `scol = list("green" = c(2, 4, 6), "blue" = 5)`, `slty = list("dashed" = c(2, 4, 6), "dotted" = 5)`, `slwd = list("2" = c(2, 4, 6), "3" = 5)`, else a named vector such as `scol = c("green" = 2, "green" = 4, "green" = 6, "blue" = 5)`, `slty = c("dashed" = 2, "dashed" = 4, "dashed" = 6, "dotted" = 5)`, `slwd = c("2" = 2, "2" = 4, "2" = 6, "3" = 5)`. The three arguments are ignored if `as.table = TRUE`.
- `new.plot, offset.x` Logical and numeric. Add to an existing plot? If so, set `new.plot = FALSE` and it is useful for the spikes to be shifted by some amount `offset.x`.
- `ymux` Numeric, y-multiplier. The response is multiplied by `ymux`. This can be useful when plotting subsets side-by-side so that the constituent proportions add up to the overall proportion.
- `...` Additional graphical arguments passed into an ordinary `plot`, for example, `xlim`, `las`, `main`.

Details

Heaping is a very commonly occurring phenomenon in retrospective self-reported survey data. Also known as *digit preference* data, it is often characterized by an excess of multiples of 10 or 5 upon rounding. For this type of data this simple function is meant to be convenient for plotting the frequencies or sample proportions of a vector `x` representing a discrete random variable. This type of plot is known as a *spike plot* in STATA circles. If `table(x)` works then this function should hopefully work. The default for `type` means that any heaping and *seeping* should easily be seen. If such features exist then *GAITD regression* is potentially useful—see `gaitdpoisson` etc. Currently missing values are ignored totally because `table(x)` is used without further arguments; this might change in the future.

Value

Returns invisibly `table(x)`.

Author(s)

T. W. Yee.

See Also

table, plot, par, deparse1, dgaitdplot, plotdgaitd, gaitdpoisson.

Examples

```
## Not run:
spikeplot(with(marital.nz, age), col = "pink2", lwd = 2)

## End(Not run)
```

---

sqrtlink *Square Root and Folded Square Root Link Functions*

Description

Computes the square root and folded square root transformations, including their inverse and their first two derivatives.

Usage

```
foldsqrtlink(theta, min = 0, max = 1, mux = sqrt(2),
     inverse = FALSE, deriv = 0, short = TRUE, tag = FALSE)
sqrtlink(theta, inverse = FALSE, deriv = 0, short = TRUE,
         tag = FALSE, c10 = c(2, -2))
```

Arguments

`theta` Numeric or character. See below for further details.

`min, max, mux` These are called L, U and K below.

`inverse, deriv, short, tag`
Details at Links.

`c10` Numeric, 2-vector `c(c1, c0)` for a linear transformation. The plain link is multiplied by `c1` and then `c0` is added so that `c1 = 1:0` is simply sqrt. The default is intended to match lcsloglink for poissonff at `lambda` (`theta`) equal to 1.

Details

The folded square root link function can be applied to parameters that lie between L and U inclusive. Numerical values of `theta` out of range result in `NA` or `NaN`.

More general information can be found at alogitlink.

Value

For `foldsqrtlink` with `deriv = 0`: $K(\sqrt{\theta - L} - \sqrt{U - \theta})$ or `mux * (sqrt(theta-min) - sqrt(max-theta))` when `inverse = FALSE`, and if `inverse = TRUE` then some more complicated function that returns a `NA` unless `theta` is between `-mux*sqrt(max-min)` and `mux*sqrt(max-min)`.

For `sqrtlink` with `deriv = 0` and `c10 = 1:0`: $\sqrt{\theta}$ when `inverse = FALSE`, and if `inverse = TRUE` then the square is returned.

For `deriv = 1`, then the function returns *d* `eta` / *d* `theta` as a function of `theta` if `inverse = FALSE`, else if `inverse = TRUE` then it returns the reciprocal.

Note

For `foldsqrtlink`, the default has, if `theta` is 0 or 1, the link function value is `-sqrt(2)` and `+sqrt(2)` respectively. These are finite values, therefore one cannot use this link function for general modelling of probabilities because of numerical problem, e.g., with `binomialff`, `cumulative`. See the example below.

Author(s)

Thomas W. Yee

See Also

`Links`, `poissonff`, `sloglink`, `hdeff`.

Examples

```
p <- seq(0.01, 0.99, by = 0.01)
foldsqrtlink(p)
max(abs(foldsqrtlink(foldsqrtlink(p), inverse = TRUE) - p))  # 0

p <- c(seq(-0.02, 0.02, by = 0.01), seq(0.97, 1.02, by = 0.01))
foldsqrtlink(p)  # Has NAs

## Not run:
p <- seq(0.01, 0.99, by = 0.01)
par(mfrow = c(2, 2), lwd = (mylwd <- 2))
y <- seq(-4, 4, length = 100)
for (d in 0:1) {
  matplot(p, cbind(   logitlink(p, deriv = d),
                   foldsqrtlink(p, deriv = d)),
          col = "blue", ylab = "transformation",
          main = ifelse(d == 0, "Some probability links",
          "First derivative"), type = "n", las = 1)
  lines(p,    logitlink(p, deriv = d), col = "green")
  lines(p,   probitlink(p, deriv = d), col = "blue")
  lines(p,  clogloglink(p, deriv = d), col = "red")
  lines(p, foldsqrtlink(p, deriv = d), col = "tan")
  if (d == 0) {
    abline(v = 0.5, h = 0, lty = "dashed")
    legend(0, 4.5, c("logitlink", "probitlink",
```

```
                      "cloglogl ink", "foldsqrtlink"),
            lwd = 2, col = c("green", "blue",
                              "red", "tan"))
  } else
    abline(v = 0.5, lty = "dashed")
}

for (d in 0) {
  matplot(y,
          cbind(   logitlink(y, deriv = d, inverse = TRUE),
                foldsqrtlink(y, deriv = d, inverse = TRUE)),
          type = "n", col = "blue", xlab = "transformation",
          ylab = "p", lwd = 2, las = 1, main = if (d == 0)
          "Some inverse probability link functions" else
          "First derivative")
  lines(y,    logitlink(y, deriv=d, inverse=TRUE), col="green")
  lines(y,   probitlink(y, deriv=d, inverse=TRUE), col="blue")
  lines(y,  cloglogl ink(y, deriv=d, inverse=TRUE), col="red")
  lines(y, foldsqrtlink(y, deriv=d, inverse=TRUE), col="tan")
  if (d == 0) {
    abline(h = 0.5, v = 0, lty = "dashed")
    legend(-4, 1, c("logitlink", "probitlink",
                    "cloglogl ink", "foldsqrtlink"), lwd = 2,
           col = c("green", "blue", "red", "tan"))
  }
}
par(lwd = 1)

## End(Not run)

# This is lucky to converge
fit.h <- vglm(agaaus ~ sm.bs(altitude),
              binomialff(foldsqrtlink(mux = 5)),
              hunua, trace = TRUE)
## Not run:
plotvgam(fit.h, se = TRUE, lcol = "orange", scol = "orange",
         main = "Orange is Hunua, Blue is Waitakere")
## End(Not run)
head(predict(fit.h, hunua, type = "response"))

## Not run:
# The following fails.
pneumo <- transform(pneumo, let = log(exposure.time))
fit <- vglm(cbind(normal, mild, severe) ~ let,
       cumulative(foldsqrtlink(mux = 10), par = TRUE, rev = TRUE),
       data = pneumo, trace = TRUE, maxit = 200)
## End(Not run)
```

---

sratio *Ordinal Regression with Stopping Ratios*

Description

Fits a stopping ratio logit/probit/cloglog/cauchit/... regression model to an ordered (preferably) factor response.

Usage

```
sratio(link = "logitlink", parallel = FALSE, reverse = FALSE,
       zero = NULL, ynames = FALSE, Thresh = NULL, Trev = reverse,
       Tref = if (Trev) "M" else 1, whitespace = FALSE)
```

Arguments

- `link`: Link function applied to the M stopping ratio probabilities. See `Links` for more choices.
- `parallel`: A logical, or formula specifying which terms have equal/unequal coefficients.
- `reverse`: Logical. By default, the stopping ratios used are $\eta_j = logit(P[Y = j|Y \geq j])$ for $j = 1, \ldots, M$. If `reverse` is `TRUE`, then $\eta_j = logit(P[Y = j{+}1|Y \leq j{+}1])$ will be used.
- `ynames`: See `multinomial` for information.
- `zero`: Can be an integer-valued vector specifying which linear/additive predictors are modelled as intercepts only. The values must be from the set {1,2,. . . ,M}. The default value means none are modelled as intercept-only terms. See `CommonVGAMffArguments` for information.
- `Thresh, Trev, Tref`: See `cumulative` for information. These arguments apply to ordinal categorical regression models.
- `whitespace`: See `CommonVGAMffArguments` for information.

Details

In this help file the response Y is assumed to be a factor with ordered values $1, 2, \ldots, M + 1$, so that M is the number of linear/additive predictors η_j.

There are a number of definitions for the *continuation ratio* in the literature. To make life easier, in the **VGAM** package, we use *continuation* ratios (see `cratio`) and *stopping* ratios. Continuation ratios deal with quantities such as `logitlink(P[Y>j|Y>=j])`.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, `rrvglm` and `vgam`.

Warning

No check is made to verify that the response is ordinal if the response is a matrix; see `ordered`.

Boersch-Supan (2021) considers a sparse data set (called `budworm`) and the numerical problems encountered when fitting models such as `cratio`, `sratio`, `cumulative`. Although improvements to links such as `clogloglink` have been made, currently these family functions have not been properly adapted to handle sparse data as well as they could.

Note

The response should be either a matrix of counts (with row sums that are all positive), or a factor. In both cases, the `y` slot returned by `vglm`/`vgam`/`rrvglm` is the matrix of counts.

For a nominal (unordered) factor response, the multinomial logit model (`multinomial`) is more appropriate.

Here is an example of the usage of the `parallel` argument. If there are covariates `x1`, `x2` and `x3`, then `parallel = TRUE ~ x1 + x2 -1` and `parallel = FALSE ~ x3` are equivalent. This would constrain the regression coefficients for `x1` and `x2` to be equal; those of the intercepts and `x3` would be different.

Author(s)

Thomas W. Yee

References

Agresti, A. (2013). *Categorical Data Analysis*, 3rd ed. Hoboken, NJ, USA: Wiley.

Boersch-Supan, P. H. (2021). Modeling insect phenology using ordinal regression and continuation ratio models. *ReScience C*, **7.1**, 1–14. doi:10.18637/jss.v032.i10.

McCullagh, P. and Nelder, J. A. (1989). *Generalized Linear Models*, 2nd ed. London: Chapman & Hall.

Tutz, G. (2012). *Regression for Categorical Data*, Cambridge: Cambridge University Press.

Yee, T. W. (2010). The **VGAM** package for categorical data analysis. *Journal of Statistical Software*, **32**, 1–34. doi:10.18637/jss.v032.i10.

See Also

`cratio`, `acat`, `cumulative`, `multinomial`, `CM.equid`, `CommonVGAMffArguments`, `margeff`, `pneumo`, `budworm`, `logitlink`, `probitlink`, `clogloglink`, `cauchitlink`.

Examples

```
pneumo <- transform(pneumo, let = log(exposure.time))
(fit <- vglm(cbind(normal, mild, severe) ~ let,
             sratio(parallel = TRUE), data = pneumo))
coef(fit, matrix = TRUE)
constraints(fit)
predict(fit)
predict(fit, untransform = TRUE)
```

step4 *Choose a model by AIC in a Stepwise Algorithm*

Description

Select a formula-based model by AIC.

Usage

```
step4(object, ...)
step4vglm(object, scope, direction = c("both", "backward", "forward"),
          trace = 1, keep = NULL, steps = 1000, k = 2, ...)
```

Arguments

`object`	an object of class "vglm". This is used as the initial model in the stepwise search.
`scope`	See `step`.
`direction`	See `step`.
`trace, keep`	See `step`.
`steps, k`	See `step`.
`...`	any additional arguments to `extractAIC.vglm`, `drop1.vglm` and `add1.vglm`.

Details

This function is a direct adaptation of `step` for `vglm-class` objects. Since `step` is not generic, the name `step4()` was adopted and it *is* generic, as well as being S4 rather than S3. It is the intent that this function should work as similar as possible to `step`.

Internally, the methods function for `vglm-class` objects calls `add1.vglm` and `drop1.vglm` repeatedly.

Value

The results are placed in the `post` slot of the stepwise-selected model that is returned. There are up to two additional components. There is an "anova" component corresponding to the steps taken in the search, as well as a "keep" component if the `keep=` argument was supplied in the call.

Warning

In general, the same warnings in `drop1.glm` and `drop1.vglm` apply here.

This function

See Also

`add1.vglm`, `drop1.vglm`, `vglm`, `trim.constraints`, `add1.glm`, `drop1.glm`, `backPain2`, `step`, `update`.

Examples

```
data("backPain2", package = "VGAM")
summary(backPain2)
fit1 <- vglm(pain ~ x2 + x3 + x4 + x2:x3 + x2:x4 + x3:x4,
             propodds, data = backPain2)
spom1 <- step4(fit1)
summary(spom1)
spom1@post$anova
```

---

studentt *Student t Distribution*

Description

Estimating the parameters of a Student t distribution.

Usage

```
studentt (ldf = "logloglink", idf = NULL, tol1 = 0.1, imethod = 1)
studentt2(df = Inf, llocation = "identitylink", lscale = "loglink",
          ilocation = NULL, iscale = NULL, imethod = 1, zero = "scale")
studentt3(llocation = "identitylink", lscale = "loglink",
          ldf = "logloglink", ilocation = NULL, iscale = NULL,
          idf = NULL, imethod = 1, zero = c("scale", "df"))
```

Arguments

`llocation, lscale, ldf`
: Parameter link functions for each parameter, e.g., for degrees of freedom ν. See `Links` for more choices. The defaults ensures the parameters are in range. A `loglog` link keeps the degrees of freedom greater than unity; see below.

`ilocation, iscale, idf`
: Optional initial values. If given, the values must be in range. The default is to compute an initial value internally.

`tol1`
: A positive value, the tolerance for testing whether an initial value is 1. Best to leave this argument alone.

`df`
: Numeric, user-specified degrees of freedom. It may be of length equal to the number of columns of a response matrix.

`imethod, zero`
: See `CommonVGAMffArguments`.

Details

The Student t density function is

$$f(y;\nu) = \frac{\Gamma((\nu+1)/2)}{\sqrt{\nu\pi}\Gamma(\nu/2)} \left(1 + \frac{y^2}{\nu}\right)^{-(\nu+1)/2}$$

for all real y. Then $E(Y) = 0$ if $\nu > 1$ (returned as the fitted values), and $Var(Y) = \nu/(\nu - 2)$ for $\nu > 2$. When $\nu = 1$ then the Student t-distribution corresponds to the standard Cauchy distribution, `cauchy1`. When $\nu = 2$ with a scale parameter of `sqrt(2)` then the Student t-distribution corresponds to the standard (Koenker) distribution, `sc.studentt2`. The degrees of freedom can be treated as a parameter to be estimated, and as a real and not an integer. The Student t distribution is used for a variety of reasons in statistics, including robust regression.

Let $Y = (T - \mu)/\sigma$ where μ and σ are the location and scale parameters respectively. Then `studentt3` estimates the location, scale and degrees of freedom parameters. And `studentt2` estimates the location, scale parameters for a user-specified degrees of freedom, `df`. And `studentt` estimates the degrees of freedom parameter only. The fitted values are the location parameters. By default the linear/additive predictors are $(\mu, \log(\sigma), \log\log(\nu))^T$ or subsets thereof.

In general convergence can be slow, especially when there are covariates.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Note

`studentt3()` and `studentt2()` can handle multiple responses.

Practical experience has shown reasonably good initial values are required. If convergence failure occurs try using arguments such as `idf`. Local solutions are also possible, especially when the degrees of freedom is close to unity or the scale parameter is close to zero.

A standard normal distribution corresponds to a *t* distribution with infinite degrees of freedom. Consequently, if the data is close to normal, there may be convergence problems; best to use `uninormal` instead.

Author(s)

T. W. Yee

References

Student (1908). The probable error of a mean. *Biometrika*, **6**, 1–25.

Zhu, D. and Galbraith, J. W. (2010). A generalized asymmetric Student-*t* distribution with application to financial econometrics. *Journal of Econometrics*, **157**, 297–305.

See Also

`uninormal`, `cauchy1`, `logistic`, `huber2`, `sc.studentt2`, `TDist`, `simulate.vlm`.

Examples

```
tdata <- data.frame(x2 = runif(nn <- 1000))
tdata <- transform(tdata, y1 = rt(nn, df = exp(exp(0.5 - x2))),
                          y2 = rt(nn, df = exp(exp(0.5 - x2))))
fit1 <- vglm(y1 ~ x2, studentt, data = tdata, trace = TRUE)
coef(fit1, matrix = TRUE)
```

```
# df inputted into studentt2() not quite right:
fit2 <- vglm(y1 ~ x2, studentt2(df = exp(exp(0.5))), tdata)
coef(fit2, matrix = TRUE)

fit3 <- vglm(cbind(y1, y2) ~ x2, studentt3, tdata, trace = TRUE)
coef(fit3, matrix = TRUE)
```

---

summary.drrvglm *Summarizing Reduced Rank Vector Generalized Linear Model (RR-VGLM) and Doubly constrained RR-VGLM Fits*

Description

These functions are all methods for class "drrvglm" or "summary.drrvglm" objects, or for class "rrvglm" or "summary.rrvglm" objects.

Usage

```
## S3 method for class 'drrvglm'
summary(object, correlation = FALSE, dispersion = NULL,
    digits = NULL, numerical = TRUE, h.step = 0.0001, omit123 = FALSE,
     omit13 = FALSE, fixA = FALSE, presid = TRUE,
    signif.stars = getOption("show.signif.stars"),
    eval0 = TRUE, nopredictors = FALSE, ...)
## S3 method for class 'summary.drrvglm'
show(x, digits = NULL,
    quote = TRUE, prefix = "", signif.stars = NULL)
## S3 method for class 'rrvglm'
summary(object, correlation = FALSE, dispersion = NULL,
    digits = NULL, numerical = TRUE, h.step = 0.0001, omit123 = FALSE,
     omit13 = FALSE, fixA = FALSE, presid = TRUE,
    signif.stars = getOption("show.signif.stars"), nopredictors = FALSE, ...)
## S3 method for class 'summary.rrvglm'
show(x, digits = NULL,
    quote = TRUE, prefix = "", signif.stars = NULL)
```

Arguments

object	an object of class "drrvglm" or "rrvglm", a result of a call to rrvglm.
x	an object of class "summary.drrvglm" or "summary.rrvglm", a result of a call to summary.drrvglm or summary.rrvglm.
dispersion	used mainly for GLMs. Not really implemented in **VGAM** so should not be used.
correlation	See summaryvglm.
digits	See summaryvglm.

`signif.stars` See `summaryvglm`.

`presid, quote` See `summaryvglm`.

`nopredictors` See `summaryvglm`.

`numerical` Logical, use a finite difference approximation for partial derivatives? If `FALSE` then theoretical formulas are used (however this option may no longer be implemented).

`h.step` Numeric, positive and close to 0. If `numerical` then this is the forward step for each finite difference approximation. That is, it plays the role of h in $(f(x+h)-f(x))/h$ for some function f. If the overall variance-covariance matrix is not positive-definite, varying this argument might make a difference.

`fixA` Logical, if `TRUE` then the largest block matrix is for **B1** and **C**, else it is for **A** and **B1**. This should not make any difference because both estimates of **B1** should be extremely similar, including the SEs.

`omit13` Logical, if `TRUE` then the (1,3) block matrix is set to **O**. That is, **A** and **C** are assumed to asymptotically uncorrelated. Setting this `TRUE` is an option when **V** (see below) is not positive-definite. If this fails, another option that is often better is to set `omit123 = TRUE`.

`omit123` Logical. If `TRUE` then *two* block matrices are set to **O** (blocks (1,2) and (1,3), else blocks (1,3) and (2,3), depending on `fixA`), This will almost surely result in an overall variance-covariance matrix that is positive-definite, however, the SEs will be biased. This argument is more extreme than `omit13`.

`prefix` See `summaryvglm`.

`eval0` Logical. Check if **V** is positive-definite? That is, all its eigenvalues are positive.

`...` Logical argument `check.2` might work here. If `TRUE` then some quantities are printed out, for checking and debugging.

Details

Most of this document concerns DRR-VGLMs but also apply equally well to RR-VGLMs as a special case.

The overall variance-covariance matrix The overall variance-covariance matrix (called **V** below) is computed. Since the parameters comprise the elements of the matrices **A**, **B1** and **C** (called here block matrices 1, 2, 3 respectively), and an alternating algorithm is used for estimation, then there are two overlapping submodels that are fitted within an IRLS algorithm. These have blocks 1 and 2, and 2 and 3, so that **B1** is common to both. They are combined into one large overall variance-covariance matrix. Argument `fixA` specifies which submodel the **B1** block is taken from. Block (1,3) is the most difficult to compute and numerical approximations based on first derivatives are used by default for this.

Sometimes the computed **V** is not positive-definite. If so, then the standard errors will be `NA`. To avoid this problem, try varying `h.step` or refitting the model with a different `Index.corner`. Argument `omit13` and `omit123` can also be used to give approximate answers. If **V** is not positive-definite then this may indicate that the model does not fit the data very well, e.g., `Rank` is not a good value. Potentially, there are many ways why the model may be ill-conditioned. Try several options and set `trace = TRUE` to monitor convergence—this is informative about how well the model and data agree.

How can one fit an ordinary RR-VGLM as a DRR-VGLM? If one uses corner constraints (default) then one should input `H.A` as a list containing `Rank` `diag(M)` matrices—one for each column of **A**. Then since `Corner = TRUE` by default, then `object@H.A.thy` has certain columns deleted due to corner constraints. The suffix `thy` stands for *theory*. In contrast, `object@H.A.alt` is the `H.A` that was inputted, and the `alt` suffix indicates the alternating algorithm.

Note that `vcov` methods exist for `rrvglm-class` and `drrvglm-class` objects.

Value

`summarydrrvglm` returns an object of class `"summary.drrvglm"`.

Warning

DRR-VGLMs are a recent development so it will take some time to get things totally ironed out. RR-VGLMs were developed a long time ago and are more well-established, however they have only recently been documented here.

Author(s)

T. W. Yee.

References

Chapter 5 of: Yee, T. W. (2015). Vector Generalized Linear and Additive Models: With an Implementation in R. New York, USA: *Springer*. Sections 5.2.2 and 5.3 are particularly relevant.

See Also

`rrvglm`, `rrvglm.control`, `vcovdrrvglm`, `CM.free`, `summaryvglm`, `summary.rrvglm-class`, `summary.drrvglm-class`.

Examples

```
  # Fit a rank-1 RR-VGLM as a DRR-VGLM.
set.seed(1); n <- 1000; S <- 6  # S must be even
myrank <- 1
rdata <- data.frame(x1 = runif(n), x2 = runif(n),
           x3 = runif(n), x4 = runif(n))
dval <- ncol(rdata)  # Number of covariates
# Involves x1, x2, ... a rank-1 model:
ymatrix <- with(rdata,
  matrix(rpois(n*S, exp(3 + x1 - 0.5*x2)), n, S))
H.C <- vector("list", dval)  # Ordinary "rrvglm"
for (i in 1:dval) H.C[[i]] <- CM.free(myrank)
names(H.C) <- paste0("x", 1:dval)
H.A <- list(CM.free(S))  # rank-1

rfit1 <- rrvglm(ymatrix ~ x1 + x2 + x3 + x4,
           poissonff, rdata, trace = TRUE)
class(rfit1)
dfit1 <- rrvglm(ymatrix ~ x1 + x2 + x3 + x4,
           poissonff, rdata, trace = TRUE,
```

```
          H.A = H.A,    # drrvglm
          H.C = H.C)    # drrvglm
class(dfit1)
Coef(rfit1)  # The RR-VGLM is the same as
Coef(dfit1)  # the DRR-VGLM.
max(abs(predict(rfit1) - predict(dfit1)))  # 0
abs(logLik(rfit1) - logLik(dfit1))  # 0
## Not run:
summary(rfit1)
summary(dfit1)

## End(Not run)
```

---

summarypvgam *Summarizing Penalized Vector Generalized Additive Model Fits*

Description

These functions are all methods for class "pvgam" or summary.pvgam objects.

Usage

```
summarypvgam(object, dispersion = NULL, digits = options()$digits - 2,
             presid = TRUE)
## S3 method for class 'summary.pvgam'
show(x, quote = TRUE, prefix = "", digits = options()$digits -
    2, signif.stars = getOption("show.signif.stars"))
```

Arguments

- `object` — an object of class "pvgam", which is the result of a call to vgam with at least one sm.os or sm.ps term.
- `x` — an object of class "summary.pvgam", which is the result of a call to summarypvgam().
- `dispersion, digits, presid` — See summaryvglm.
- `quote, prefix, signif.stars` — See summaryvglm.

Details

This methods function reports a summary more similar to summary.gam from **mgcv** than summary.gam() from **gam**. It applies to G2-VGAMs using sm.os and O-splines, else sm.ps and P-splines. In particular, the hypothesis test for whether each sm.os or sm.ps term can be deleted follows quite closely to summary.gam. The p-values from this type of test tend to be biased downwards (too small) and corresponds to p.type = 5. It is hoped in the short future that improved p-values be implemented, somewhat like the default of summary.gam. This methods function was adapted from summary.gam.

Value

summarypvgam returns an object of class "summary.pvgam"; see summary.pvgam-class.

Warning

See sm.os.

See Also

vgam, summaryvgam, summary.pvgam-class, sm.os, sm.ps, summary.glm, summary.lm, summary.gam from **mgcv**, summaryvgam for G1-VGAMs.

Examples

```
hfit2 <- vgam(agaaus ~ sm.os(altitude), binomialff, data = hunua)
coef(hfit2, matrix = TRUE)
summary(hfit2)
```

---

summaryvgam *Summarizing Vector Generalized Additive Model Fits*

Description

These functions are all methods for class vgam or summary.vgam objects.

Usage

```
summaryvgam(object, dispersion = NULL, digits = options()$digits - 2,
            presid = TRUE, nopredictors = FALSE)
## S3 method for class 'summary.vgam'
show(x, quote = TRUE, prefix = "",
                        digits = options()$digits-2, nopredictors = NULL)
```

Arguments

object	an object of class "vgam", which is the result of a call to vgam with at least one s term.
x	an object of class "summary.vgam", which is the result of a call to summaryvgam().
dispersion, digits, presid	See summaryvglm.
quote, prefix, nopredictors	See summaryvglm.

Details

This methods function reports a summary more similar to summary.gam() from **gam** than summary.gam from **mgcv**. It applies to G1-VGAMs using s and vector backfitting. In particular, an approximate score test for *linearity* is conducted for each s term—see Section 4.3.4 of Yee (2015) for details. The p-values from this type of test tend to be biased upwards (too large).

Value

summaryvgam returns an object of class "summary.vgam"; see summary.vgam-class.

See Also

vgam, summary.glm, summary.lm, summary.gam from **mgcv**, summarypvgam for P-VGAMs.

Examples

```
hfit <- vgam(agaaus ~ s(altitude, df = 2), binomialff, data = hunua)
summary(hfit)
summary(hfit)@anova  # Table for (approximate) testing of linearity
```

---

summaryvglm *Summarizing Vector Generalized Linear Model Fits*

Description

These functions are all methods for class vglm or summary.vglm objects.

Usage

```
summaryvglm(object, correlation = FALSE, dispersion = NULL,
            digits = NULL, presid = FALSE,
            HDEtest = TRUE, hde.NA = TRUE, threshold.hde = 0.001,
            signif.stars = getOption("show.signif.stars"),
            nopredictors = FALSE,
            lrt0.arg = FALSE, score0.arg = FALSE, wald0.arg = FALSE,
            values0 = 0, subset = NULL, omit1s = TRUE,
            ...)
## S3 method for class 'summary.vglm'
show(x, digits = max(3L, getOption("digits") - 3L),
           quote = TRUE, prefix = "", presid = length(x@pearson.resid) > 0,
           HDEtest = TRUE, hde.NA = TRUE, threshold.hde = 0.001,
           signif.stars = NULL, nopredictors = NULL,
           top.half.only = FALSE, ...)
```

Arguments

object	an object of class "vglm", usually, a result of a call to vglm.
x	an object of class "summary.vglm", usually, a result of a call to summaryvglm().
dispersion	used mainly for GLMs. See summary.glm.
correlation	logical; if TRUE, the correlation matrix of the estimated parameters is returned and printed.
digits	the number of significant digits to use when printing.
signif.stars	logical; if TRUE, 'significance stars' are printed for each coefficient.

presid — Pearson residuals; print out some summary statistics of these?

HDEtest — logical; if TRUE (the default) then a test for the HDE is performed, else all arguments related to the HDE are ignored.

hde.NA — logical; if a test for the Hauck-Donner effect is done (for each coefficient) and it is affirmative should that Wald test p-value be replaced by an NA? The default is to do so. Setting hde.NA = FALSE will print the p-value even though it will be biased upwards. Also see argument threshold.hde.

threshold.hde — numeric; used if hde.NA = TRUE and is present for some coefficients. Only p-values greater than this argument will be replaced by an NA, the reason being that small p-values will already be statistically significant. Hence setting threshold.hde = 0 will print out a NA if the HDE is present.

quote — Fed into print().

nopredictors — logical; if TRUE the names of the linear predictors are not printed out. The default is that they are.

lrt0.arg, score0.arg, wald0.arg — Logical. If lrt0.arg = TRUE then the other arguments are passed into lrt.stat.vlm and the equivalent of the so-called Wald table is outputted. Similarly, if score0.arg = TRUE then the other arguments are passed into score.stat.vlm and the equivalent of the so-called Wald table is outputted. Similarly, if wald0.arg = TRUE then the other arguments are passed into wald.stat.vlm and the Wald table corresponding to that is outputted. See details below. Setting any of these will result in further IRLS iterations being performed, therefore may be computationally expensive.

values0, subset, omit1s — These arguments are used if any of the lrt0.arg, score0.arg, wald0.arg arguments are used. They are passed into the appropriate function, such as wald.stat.vlm.

top.half.only — logical; if TRUE then only print out the top half of the usual output. Used for P-VGAMs.

prefix — Not used.

... — Not used.

Details

Originally, summaryvglm() was written to be very similar to summary.glm, however now there are a quite a few more options available. By default, show.summary.vglm() tries to be smart about formatting the coefficients, standard errors, etc. and additionally gives 'significance stars' if signif.stars is TRUE. The coefficients component of the result gives the estimated coefficients and their estimated standard errors, together with their ratio. This third column is labelled z value regardless of whether the dispersion is estimated or known (or fixed by the family). A fourth column gives the two-tailed p-value corresponding to the z ratio based on a Normal reference distribution.

In general, the t distribution is not used, but the normal distribution is.

Correlations are printed to two decimal places (or symbolically): to see the actual correlations print summary(object)@correlation directly.

The Hauck-Donner effect (HDE) is tested for almost all models; see hdeff.vglm for details. Arguments hde.NA and threshold.hde here are meant to give some control of the output if this aberration of the Wald statistic occurs (so that the p-value is biased upwards). If the HDE is present then using lrt.stat.vlm to get a more accurate p-value is a good alternative as p-values based on the likelihood ratio test (LRT) tend to be more accurate than Wald tests and do not suffer from the HDE. Alternatively, if the HDE is present then using wald0.arg = TRUE will compute Wald statistics that are HDE-free; see wald.stat.

The arguments lrt0.arg and score0.arg enable the so-called Wald table to be replaced by the equivalent LRT and Rao score test table; see lrt.stat.vlm, score.stat. Further IRLS iterations are performed for both of these, hence the computational cost might be significant.

It is possible for programmers to write a methods function to print out extra quantities when summary(vglmObject) is called. The generic function is summaryvglmS4VGAM(), and one can use the S4 function setMethod to compute the quantities needed. Also needed is the generic function is showsummaryvglmS4VGAM() to actually print the quantities out.

Value

summaryvglm returns an object of class "summary.vglm"; see summary.vglm-class.

Warning

Currently the SE column is deleted when lrt0 = TRUE because SEs are not so meaningful with the LRT. In the future an SE column may be inserted (with NA values) so that it has 4-column output like the other tests. In the meantime, the columns of this matrix should be accessed by name and not number.

Author(s)

T. W. Yee.

See Also

vglm, confintvglm, vcovvlm, summary.rrvglm, summary.glm, summary.lm, summary, hdeff.vglm, lrt.stat.vlm, score.stat, wald.stat.

Examples

```
## For examples see example(glm)
pneumo <- transform(pneumo, let = log(exposure.time))
(afit <- vglm(cbind(normal, mild, severe) ~ let, acat, data = pneumo))
coef(afit, matrix = TRUE)
summary(afit)  # Might suffer from the Hauck-Donner effect
coef(summary(afit))
summary(afit, lrt0 = TRUE, score0 = TRUE, wald0 = TRUE)
```

SURff *Seemingly Unrelated Regressions Family Function*

Description

Fits a system of seemingly unrelated regressions.

Usage

```
SURff(mle.normal = FALSE,
      divisor = c("n", "n-max(pj,pk)", "sqrt((n-pj)*(n-pk))"),
      parallel = FALSE, Varcov = NULL, matrix.arg = FALSE)
```

Arguments

mle.normal	Logical. If TRUE then the MLE, assuming multivariate normal errors, is computed; the effect is just to add a loglikelihood slot to the returned object. Then it results in the *maximum likelihood estimator*.
divisor	Character, partial matching allowed and the first choice is the default. The divisor for the estimate of the covariances. If "n" then the estimate will be biased. If the others then the estimate will be unbiased for some elements. If mle.normal = TRUE and this argument is not "n" then a warning or an error will result.
parallel	See CommonVGAMffArguments. If parallel = TRUE then the constraint applies to the intercept too.
Varcov	Numeric. This may be assigned a variance-covariance of the errors. If matrix.arg then this is a $M \times M$ matrix. If !matrix.arg then this is a $M \times M$ matrix in matrix-band format (a vector with at least M and at most M*(M+1)/2 elements).
matrix.arg	Logical. Of single length.

Details

Proposed by Zellner (1962), the basic seemingly unrelated regressions (SUR) model is a set of LMs ($M > 1$ of them) tied together at the error term level. Each LM's model matrix may potentially have its own set of predictor variables.

Zellner's efficient (ZEF) estimator (also known as *Zellner's two-stage Aitken estimator*) can be obtained by setting maxit = 1 (and possibly divisor = "sqrt" or divisor = "n-max").

The default value of maxit (in vglm.control) probably means *iterative GLS* (IGLS) estimator is computed because IRLS will probably iterate to convergence. IGLS means, at each iteration, the residuals are used to estimate the error variance-covariance matrix, and then the matrix is used in the GLS. The IGLS estimator is also known as *Zellner's iterative Aitken estimator*, or IZEF.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm and vgam.

Warning

The default convergence criterion may be a little loose. Try setting `epsilon = 1e-11`, especially with `mle.normal = TRUE`.

Note

The fitted object has slot `@extra$ncols.X.lm` which is a M vector with the number of parameters for each LM. Also, `@misc$values.divisor` is the M-vector of `divisor` values.

Constraint matrices are needed in order to specify which response variables that each term on the RHS of the formula is a regressor for. See the `constraints` argument of `vglm` for more information.

Author(s)

T. W. Yee.

References

Zellner, A. (1962). An Efficient Method of Estimating Seemingly Unrelated Regressions and Tests for Aggregation Bias. *J. Amer. Statist. Assoc.*, **57**(298), 348–368.

Kmenta, J. and Gilbert, R. F. (1968). Small Sample Properties of Alternative Estimators of Seemingly Unrelated Regressions. *J. Amer. Statist. Assoc.*, **63**(324), 1180–1200.

See Also

`uninormal`, `gew`.

Examples

```
# Obtain some of the results of p.1199 of Kmenta and Gilbert (1968)
clist <- list("(Intercept)" = diag(2),
              "capital.g"   = rbind(1, 0),
              "value.g"     = rbind(1, 0),
              "capital.w"   = rbind(0, 1),
              "value.w"     = rbind(0, 1))
zef1 <- vglm(cbind(invest.g, invest.w) ~
             capital.g + value.g + capital.w + value.w,
             SURff(divisor = "sqrt"), maxit = 1,
             data = gew, trace = TRUE, constraints = clist)

round(coef(zef1, matrix = TRUE), digits = 4)  # ZEF
zef1@extra$ncols.X.lm
zef1@misc$divisor
zef1@misc$values.divisor
round(sqrt(diag(vcov(zef1))),    digits = 4)  # SEs
nobs(zef1, type = "lm")
df.residual(zef1, type = "lm")


mle1 <- vglm(cbind(invest.g, invest.w) ~
             capital.g + value.g + capital.w + value.w,
```

```
              SURff(mle.normal = TRUE),
              epsilon = 1e-11,
              data = gew, trace = TRUE, constraints = clist)
round(coef(mle1, matrix = TRUE), digits = 4)  # MLE
round(sqrt(diag(vcov(mle1))),    digits = 4)  # SEs
```

SurvS4 *Create a Survival Object*

Description

Create a survival object, usually used as a response variable in a model formula.

Usage

```
SurvS4(time, time2, event, type =, origin = 0)
is.SurvS4(x)
```

Arguments

`time`	for right censored data, this is the follow up time. For interval data, the first argument is the starting time for the interval.
`x`	any R object.
`event`	The status indicator, normally 0=alive, 1=dead. Other choices are `TRUE`/`FALSE` (`TRUE` = death) or 1/2 (2=death). For interval censored data, the status indicator is 0=right censored, 1=event at `time`, 2=left censored, 3=interval censored. Although unusual, the event indicator can be omitted, in which case all subjects are assumed to have an event.
`time2`	ending time of the interval for interval censored or counting process data only. Intervals are assumed to be open on the left and closed on the right, `(start, end]`. For counting process data, `event` indicates whether an event occurred at the end of the interval.
`type`	character string specifying the type of censoring. Possible values are `"right"`, `"left"`, `"counting"`, `"interval"`, or `"interval2"`. The default is `"right"` or `"counting"` depending on whether the `time2` argument is absent or present, respectively.
`origin`	for counting process data, the hazard function origin. This is most often used in conjunction with a model containing time dependent strata in order to align the subjects properly when they cross over from one strata to another.

Details

Typical usages are

```
SurvS4(time, event)
SurvS4(time, time2, event, type=, origin=0)
```

In theory it is possible to represent interval censored data without a third column containing the explicit status. Exact, right censored, left censored and interval censored observation would be represented as intervals of (a,a), (a, infinity), (-infinity,b), and (a,b) respectively; each specifying the interval within which the event is known to have occurred.

If `type = "interval2"` then the representation given above is assumed, with NA taking the place of infinity. If 'type="interval" `event` must be given. If `event` is `0`, `1`, or `2`, the relevant information is assumed to be contained in `time`, the value in `time2` is ignored, and the second column of the result will contain a placeholder.

Presently, the only methods allowing interval censored data are the parametric models computed by `survreg`, so the distinction between open and closed intervals is unimportant. The distinction is important for counting process data and the Cox model.

The function tries to distinguish between the use of 0/1 and 1/2 coding for left and right censored data using `if (max(status)==2)`. If 1/2 coding is used and all the subjects are censored, it will guess wrong. Use 0/1 coding in this case.

Value

An object of class `SurvS4` (formerly `Surv`). There are methods for `print`, `is.na`, and subscripting survival objects. `SurvS4` objects are implemented as a matrix of 2 or 3 columns.

In the case of `is.SurvS4`, a logical value `TRUE` if `x` inherits from class `"SurvS4"`, otherwise a `FALSE`.

Note

The purpose of having `SurvS4` in **VGAM** is so that the same input can be fed into `vglm` as functions in **survival** such as `survreg`. The class name has been changed from `"Surv"` to `"SurvS4"`; see `SurvS4-class`.

The format `J+` is interpreted in **VGAM** as $\geq J$. If `type="interval"` then these should not be used in **VGAM**: `(L,U-]` or `(L,U+]`.

Author(s)

The code and documentation comes from **survival**. Slight modifications have been made for conversion to S4 by T. W. Yee. Also, for `"interval"` data, `as.character.SurvS4()` has been modified to print intervals of the form `(start, end]` and not `[start, end]` as previously. (This makes a difference for discrete data, such as for `cens.poisson`). All **VGAM** family functions beginning with `"cen"` require the packaging function `Surv` to format the input.

See Also

`SurvS4-class`, `cens.poisson`, `survreg`, `leukemia`.

Examples

```
with(leukemia, SurvS4(time, status))
class(with(leukemia, SurvS4(time, status)))
```

SurvS4-class *Class "SurvS4"*

Description

S4 version of the Surv class.

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class "Surv", directly. Class "matrix", directly. Class "oldClass", by class "Surv", distance 2. Class "structure", by class "matrix", distance 2. Class "array", by class "matrix", distance 2. Class "vector", by class "matrix", distance 3, with explicit coerce. Class "vector", by class "matrix", distance 4, with explicit coerce.

Methods

show `signature(object = "SurvS4")`: ...

Warning

This code has not been thoroughly tested.

Note

The purpose of having `SurvS4` in **VGAM** is so that the same input can be fed into `vglm` as functions in **survival** such as `survreg`.

Author(s)

T. W. Yee.

References

See **survival**.

See Also

`SurvS4`.

Examples

```
showClass("SurvS4")
```

TIC *Takeuchi's Information Criterion*

Description

Calculates the Takeuchi information criterion for a fitted model object for which a log-likelihood value has been obtained.

Usage

```
TIC(object, ...)
TICvlm(object, ...)
```

Arguments

`object`	A **VGAM** object having class `vglm-class`.
`...`	Other possible arguments fed into `logLik` in order to compute the log-likelihood.

Details

The following formula is used for VGLMs: -2log-likelihood$+2trace(VK)$, where V is the inverse of the EIM from the fitted model, and K is the outer product of the score vectors. Both V and K are order-$p.VLM$ matrices. One has V equal to `vcov(object)`, and K is computed by taking the outer product of the output from the `deriv` slot multiplied by the large VLM matrix and then taking their sum. Hence for the huge majority of models, the penalty is computed at the MLE and is empirical in nature. Theoretically, if the fitted model is the true model then AIC equals TIC.

When there are prior weights the score vectors are divided by the square root of these, because $(a_i U_i/\sqrt{a_i})^2 = a_i U_i^2$.

This code relies on the log-likelihood being defined, and computed, for the object. When comparing fitted objects, the smaller the TIC, the better the fit. The log-likelihood and hence the TIC is only defined up to an additive constant.

Currently any estimated scale parameter (in GLM parlance) is ignored by treating its value as unity. Also, currently this function is written only for `vglm` objects and not `vgam` or `rrvglm`, etc., objects.

Value

Returns a numeric TIC value.

Warning

This code has not been double-checked. The general applicability of `TIC` for the VGLM/VGAM classes has not been developed fully. In particular, `TIC` should not be run on some **VGAM** family functions because of violation of certain regularity conditions, etc.

Some authors note that quite large sample sizes are needed for this IC to work reasonably well.

Note

TIC has not been defined for RR-VGLMs, QRR-VGLMs, etc., yet.

See `AICvlm` about models such as `posbernoulli.tb` that require `posbinomial(omit.constant = TRUE)`.

Author(s)

T. W. Yee.

References

Takeuchi, K. (1976). Distribution of informational statistics and a criterion of model fitting. (In Japanese). *Suri-Kagaku* (Mathematic Sciences), **153**, 12–18.

Burnham, K. P. and Anderson, D. R. (2002). *Model Selection and Multi-Model Inference: A Practical Information-Theoretic Approach*, 2nd ed. New York, USA: Springer.

See Also

VGLMs are described in `vglm-class`; `AIC`, `AICvlm`. `BICvlm`.

Examples

```
pneumo <- transform(pneumo, let = log(exposure.time))
(fit1 <- vglm(cbind(normal, mild, severe) ~ let,
              cumulative(parallel = TRUE, reverse = TRUE), data = pneumo))
coef(fit1, matrix = TRUE)
TIC(fit1)
(fit2 <- vglm(cbind(normal, mild, severe) ~ let,
              cumulative(parallel = FALSE, reverse = TRUE), data = pneumo))
coef(fit2, matrix = TRUE)
TIC(fit2)
```

`Tobit` *The Tobit Distribution*

Description

Density, distribution function, quantile function and random generation for the Tobit model.

Usage

```
dtobit(x, mean = 0, sd = 1, Lower = 0, Upper = Inf, log = FALSE)
ptobit(q, mean = 0, sd = 1, Lower = 0, Upper = Inf,
       lower.tail = TRUE, log.p = FALSE)
qtobit(p, mean = 0, sd = 1, Lower = 0, Upper = Inf,
       lower.tail = TRUE, log.p = FALSE)
rtobit(n, mean = 0, sd = 1, Lower = 0, Upper = Inf)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. If `length(n) > 1` then the length is taken to be the number required.
`Lower, Upper`	vector of lower and upper thresholds.
`mean, sd, lower.tail, log, log.p`	see `rnorm`.

Details

See `tobit`, the **VGAM** family function for estimating the parameters, for details. Note that the density at `Lower` and `Upper` is the the area to the left and right of those points. Thus there are two spikes (but less in value); see the example below. Consequently, `dtobit(Lower) + dtobit(Upper) +` the area in between equals unity.

Value

`dtobit` gives the density, `ptobit` gives the distribution function, `qtobit` gives the quantile function, and `rtobit` generates random deviates.

Author(s)

T. W. Yee

See Also

`tobit`, `rnorm`.

Examples

```
mu <- 0.5; x <- seq(-2, 4, by = 0.01)
Lower <- -1; Upper <- 2.0

integrate(dtobit, lower = Lower, upper = Upper,
          mean = mu, Lower = Lower, Upper = Upper)$value +
dtobit(Lower, mean = mu, Lower = Lower, Upper = Upper) +
dtobit(Upper, mean = mu, Lower = Lower, Upper = Upper)  # Adds to 1

## Not run:
plot(x, ptobit(x, m = mu, Lower = Lower, Upper = Upper),
     type = "l", ylim = 0:1, las = 1, col = "orange",
     ylab = paste("ptobit(m = ", mu, ", sd = 1, Lower =", Lower,
                  ", Upper =", Upper, ")"),
     main = "Orange is the CDF; blue is density",
     sub = "Purple lines are the 10,20,...,90 percentiles")
abline(h = 0)
lines(x, dtobit(x, m = mu, L = Lower, U = Upper), col = "blue")
```

```
probs <- seq(0.1, 0.9, by = 0.1)
Q <- qtobit(probs, m = mu, Lower = Lower, Upper = Upper)
lines(Q, ptobit(Q, m = mu, Lower = Lower, Upper = Upper),
      col = "purple", lty = "dashed", type = "h")
lines(Q, dtobit(Q, m = mu, Lower = Lower, Upper = Upper),
      col = "darkgreen", lty = "dashed", type = "h")
abline(h = probs, col = "purple", lty = "dashed")
max(abs(ptobit(Q, mu, L = Lower, U = Upper) - probs))  # Should be 0

epts <- c(Lower, Upper)  # Endpoints have a spike (not quite, actually)
lines(epts, dtobit(epts, m = mu, Lower = Lower, Upper = Upper),
      col = "blue", lwd = 3, type = "h")

## End(Not run)
```

---

tobit *Tobit Regression*

Description

Fits a Tobit regression model.

Usage

```
tobit(Lower = 0, Upper = Inf, lmu = "identitylink",
      lsd = "loglink", imu = NULL, isd = NULL,
      type.fitted = c("uncensored", "censored", "mean.obs"),
      byrow.arg = FALSE, imethod = 1, zero = "sd")
```

Arguments

`Lower` Numeric. It is the value L described below. Any value of the linear model $x_i^T \beta$ that is less than this lowerbound is assigned this value. Hence this should be the smallest possible value in the response variable. May be a vector (see below for more information).

`Upper` Numeric. It is the value U described below. Any value of the linear model $x_i^T \beta$ that is greater than this upperbound is assigned this value. Hence this should be the largest possible value in the response variable. May be a vector (see below for more information).

`lmu, lsd` Parameter link functions for the mean and standard deviation parameters. See `Links` for more choices. The standard deviation is a positive quantity, therefore a log link is its default.

`imu, isd, byrow.arg` See `CommonVGAMffArguments` for information.

`type.fitted` Type of fitted value returned. The first choice is default and is the ordinary uncensored or unbounded linear model. If `"censored"` then the fitted values in the interval $[L, U]$. If `"mean.obs"` then the mean of the observations is returned;

this is a doubly truncated normal distribution augmented by point masses at the truncation points (see dtobit). See CommonVGAMffArguments for more information.

imethod
: Initialization method. Either 1 or 2 or 3, this specifies some methods for obtaining initial values for the parameters. See CommonVGAMffArguments for information.

zero
: A vector, e.g., containing the value 1 or 2. If so, the mean or standard deviation respectively are modelled as an intercept-only. Setting zero = NULL means both linear/additive predictors are modelled as functions of the explanatory variables. See CommonVGAMffArguments for more information.

Details

The Tobit model can be written

$$y_i^* = x_i^T \beta + \varepsilon_i$$

where the $e_i \sim N(0, \sigma^2)$ independently and $i = 1, \ldots, n$. However, we measure $y_i = y_i^*$ only if $y_i^* > L$ and $y_i^* < U$ for some cutpoints L and U. Otherwise we let $y_i = L$ or $y_i = U$, whatever is closer. The Tobit model is thus a multiple linear regression but with censored responses if it is below or above certain cutpoints.

The defaults for Lower and Upper and lmu correspond to the *standard* Tobit model. Fisher scoring is used for the standard and nonstandard models. By default, the mean $x_i^T \beta$ is the first linear/additive predictor, and the log of the standard deviation is the second linear/additive predictor. The Fisher information matrix for uncensored data is diagonal. The fitted values are the estimates of $x_i^T \beta$.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Warning

If values of the response and Lower and/or Upper are not integers then there is the danger that the value is wrongly interpreted as uncensored. For example, if the first 10 values of the response were runif(10) and Lower was assigned these value then testing y[1:10] == Lower[1:10] is numerically fraught. Currently, if any y < Lower or y > Upper then a warning is issued. The function round2 may be useful.

Note

The response can be a matrix. If so, then Lower and Upper are recycled into a matrix with the number of columns equal to the number of responses, and the recycling is done row-wise *if* byrow.arg = TRUE. The default order is as matrix, which is byrow.arg = FALSE. For example, these are returned in fit4@misc$Lower and fit4@misc$Upper below.

If there is no censoring then uninormal is recommended instead. Any value of the response less than Lower or greater than Upper will be assigned the value Lower and Upper respectively, and a warning will be issued. The fitted object has components censoredL and censoredU in the extra slot which specifies whether observations are censored in that direction. The function cens.normal is an alternative to tobit().

When obtaining initial values, if the algorithm would otherwise want to fit an underdetermined system of equations, then it uses the entire data set instead. This might result in rather poor quality initial values, and consequently, monitoring convergence is advised.

Author(s)

Thomas W. Yee

References

Tobin, J. (1958). Estimation of relationships for limited dependent variables. *Econometrica* **26**, 24–36.

See Also

`rtobit`, `cens.normal`, `uninormal`, `double.cens.normal`, `posnormal`, `CommonVGAMffArguments`, `round2`, `mills.ratio`, `margeff`, `rnorm`.

Examples

```
# Here, fit1 is a standard Tobit model and fit2 is nonstandard
tdata <- data.frame(x2 = seq(-1, 1, length = (nn <- 100)))
set.seed(1)
Lower <- 1; Upper <- 4  # For the nonstandard Tobit model
tdata <- transform(tdata,
                   Lower.vec = rnorm(nn, Lower, 0.5),
                   Upper.vec = rnorm(nn, Upper, 0.5))
meanfun1 <- function(x) 0 + 2*x
meanfun2 <- function(x) 2 + 2*x
meanfun3 <- function(x) 3 + 2*x
tdata <- transform(tdata,
  y1 = rtobit(nn, mean = meanfun1(x2)),  # Standard Tobit model
  y2 = rtobit(nn, mean = meanfun2(x2), Lower = Lower, Upper = Upper),
  y3 = rtobit(nn, mean = meanfun3(x2), Lower = Lower.vec,
              Upper = Upper.vec),
  y4 = rtobit(nn, mean = meanfun3(x2), Lower = Lower.vec,
              Upper = Upper.vec))
with(tdata, table(y1 == 0))  # How many censored values?
with(tdata, table(y2 == Lower | y2 == Upper))  # Ditto
with(tdata, table(attr(y2, "cenL")))
with(tdata, table(attr(y2, "cenU")))

fit1 <- vglm(y1 ~ x2, tobit, data = tdata, trace = TRUE)
coef(fit1, matrix = TRUE)
summary(fit1)

fit2 <- vglm(y2 ~ x2,
             tobit(Lower = Lower, Upper = Upper, type.f = "cens"),
             data = tdata, trace = TRUE)
table(fit2@extra$censoredL)
table(fit2@extra$censoredU)
coef(fit2, matrix = TRUE)
```

```
fit3 <- vglm(y3 ~ x2, tobit(Lower = with(tdata, Lower.vec),
                            Upper = with(tdata, Upper.vec),
                            type.f = "cens"),
             data = tdata, trace = TRUE)
table(fit3@extra$censoredL)
table(fit3@extra$censoredU)
coef(fit3, matrix = TRUE)

# fit4 is fit3 but with type.fitted = "uncen".
fit4 <- vglm(cbind(y3, y4) ~ x2,
             tobit(Lower = rep(with(tdata, Lower.vec), each = 2),
                   Upper = rep(with(tdata, Upper.vec), each = 2),
                   byrow.arg = TRUE),
             data = tdata, crit = "coeff", trace = TRUE)
head(fit4@extra$censoredL)  # A matrix
head(fit4@extra$censoredU)  # A matrix
head(fit4@misc$Lower)       # A matrix
head(fit4@misc$Upper)       # A matrix
coef(fit4, matrix = TRUE)

## Not run:  # Plot fit1--fit4
par(mfrow = c(2, 2))

plot(y1 ~ x2, tdata, las = 1, main = "Standard Tobit model",
     col = as.numeric(attr(y1, "cenL")) + 3,
     pch = as.numeric(attr(y1, "cenL")) + 1)
legend(x = "topleft", leg = c("censored", "uncensored"),
       pch = c(2, 1), col = c("blue", "green"))
legend(-1.0, 2.5, c("Truth", "Estimate", "Naive"), lwd = 2,
       col = c("purple", "orange", "black"), lty = c(1, 2, 2))
lines(meanfun1(x2) ~ x2, tdata, col = "purple", lwd = 2)
lines(fitted(fit1) ~ x2, tdata, col = "orange", lwd = 2, lty = 2)
lines(fitted(lm(y1 ~ x2, tdata)) ~ x2, tdata, col = "black",
      lty = 2, lwd = 2)  # This is simplest but wrong!

plot(y2 ~ x2, data = tdata, las = 1, main = "Tobit model",
     col = as.numeric(attr(y2, "cenL")) + 3 +
           as.numeric(attr(y2, "cenU")),
     pch = as.numeric(attr(y2, "cenL")) + 1 +
           as.numeric(attr(y2, "cenU")))
legend(x = "topleft", leg = c("censored", "uncensored"),
       pch = c(2, 1), col = c("blue", "green"))
legend(-1.0, 3.5, c("Truth", "Estimate", "Naive"), lwd = 2,
       col = c("purple", "orange", "black"), lty = c(1, 2, 2))
lines(meanfun2(x2) ~ x2, tdata, col = "purple", lwd = 2)
lines(fitted(fit2) ~ x2, tdata, col = "orange", lwd = 2, lty = 2)
lines(fitted(lm(y2 ~ x2, tdata)) ~ x2, tdata, col = "black",
      lty = 2, lwd = 2)  # This is simplest but wrong!

plot(y3 ~ x2, data = tdata, las = 1,
     main = "Tobit model with nonconstant censor levels",
     col = as.numeric(attr(y3, "cenL")) + 2 +
```

```
            as.numeric(attr(y3, "cenU") * 2),
      pch = as.numeric(attr(y3, "cenL")) + 1 +
            as.numeric(attr(y3, "cenU") * 2))
legend(x = "topleft", pch = c(2, 3, 1), col = c(3, 4, 2),
       leg = c("censoredL", "censoredU", "uncensored"))
legend(-1.0, 3.5, c("Truth", "Estimate", "Naive"), lwd = 2,
       col = c("purple", "orange", "black"), lty = c(1, 2, 2))
lines(meanfun3(x2) ~ x2, tdata, col = "purple", lwd = 2)
lines(fitted(fit3) ~ x2, tdata, col = "orange", lwd = 2, lty = 2)
lines(fitted(lm(y3 ~ x2, tdata)) ~ x2, tdata, col = "black",
      lty = 2, lwd = 2)  # This is simplest but wrong!

plot(y3 ~ x2, data = tdata, las = 1,
     main = "Tobit model with nonconstant censor levels",
     col = as.numeric(attr(y3, "cenL")) + 2 +
           as.numeric(attr(y3, "cenU") * 2),
     pch = as.numeric(attr(y3, "cenL")) + 1 +
           as.numeric(attr(y3, "cenU") * 2))
legend(x = "topleft", pch = c(2, 3, 1), col = c(3, 4, 2),
       leg = c("censoredL", "censoredU", "uncensored"))
legend(-1.0, 3.5, c("Truth", "Estimate", "Naive"), lwd = 2,
       col = c("purple", "orange", "black"), lty = c(1, 2, 2))
lines(meanfun3(x2) ~ x2, data = tdata, col = "purple", lwd = 2)
lines(fitted(fit4)[, 1] ~ x2, tdata, col="orange", lwd = 2, lty = 2)
lines(fitted(lm(y3 ~ x2, tdata)) ~ x2, data = tdata, col = "black",
      lty = 2, lwd = 2)  # This is simplest but wrong!

## End(Not run)
```

Tol *Tolerances*

Description

Generic function for the *tolerances* of a model.

Usage

```
Tol(object, ...)
```

Arguments

object	An object for which the computation or extraction of a tolerance or tolerances is meaningful.
...	Other arguments fed into the specific methods function of the model. Sometimes they are fed into the methods function for Coef.

Details

Different models can define an optimum in different ways. Many models have no such notion or definition.

Tolerances occur in quadratic ordination, i.e., CQO and UQO. They have ecological meaning because a high tolerance for a species means the species can survive over a large environmental range (stenoecous species), whereas a small tolerance means the species' niche is small (eurycous species). Mathematically, the tolerance is like the variance of a normal distribution.

Value

The value returned depends specifically on the methods function invoked. For a `cqo` binomial or Poisson fit, this function returns a $R \times R \times S$ array, where R is the rank and S is the number of species. Each tolerance matrix ought to be positive-definite, and for a rank-1 fit, taking the square root of each tolerance matrix results in each species' tolerance (like a standard deviation).

Warning

There is a direct inverse relationship between the scaling of the latent variables (site scores) and the tolerances. One normalization is for the latent variables to have unit variance. Another normalization is for all the tolerances to be unit. These two normalization cannot simultaneously hold in general. For rank-*R>1* models it becomes more complicated because the latent variables are also uncorrelated. An important argument when fitting quadratic ordination models is whether `eq.tolerances` is `TRUE` or `FALSE`. See Yee (2004) for details.

Note

Tolerances are undefined for 'linear' and additive ordination models. They are well-defined for quadratic ordination models.

Author(s)

Thomas W. Yee

References

Yee, T. W. (2004). A new technique for maximum-likelihood canonical Gaussian ordination. *Ecological Monographs*, **74**, 685–701.

Yee, T. W. (2006). Constrained additive ordination. *Ecology*, **87**, 203–213.

See Also

`Tol.qrrvglm`. `Max`, `Opt`, `cqo`, `rcim` for UQO.

Examples

```
## Not run:
set.seed(111)  # This leads to the global solution
hspider[,1:6] <- scale(hspider[, 1:6])  # Standardized environmental vars
p1 <- cqo(cbind(Alopacce, Alopcune, Alopfabr, Arctlute, Arctperi,
```

```
                 Auloalbi, Pardlugu, Pardmont, Pardnigr, Pardpull,
                 Trocterr, Zoraspin) ~
           WaterCon + BareSand + FallTwig + CoveMoss + CoveHerb + ReflLux,
           poissonff, data = hspider, Crow1positive = FALSE)

Tol(p1)

## End(Not run)
```

Topple *The Topp-Leone Distribution*

Description

Density, distribution function, quantile function and random generation for the Topp-Leone distribution.

Usage

```
dtopple(x, shape, log = FALSE)
ptopple(q, shape, lower.tail = TRUE, log.p = FALSE)
qtopple(p, shape)
rtopple(n, shape)
```

Arguments

`x, q, p, n`	Same as `Uniform`.
`shape`	the (shape) parameter, which lies in $(0, 1)$.
`log`	Logical. If `log = TRUE` then the logarithm of the density is returned.
`lower.tail, log.p`	Same meaning as in `pnorm` or `qnorm`.

Details

See `topple`, the **VGAM** family function for estimating the (shape) parameter s by maximum likelihood estimation, for the formula of the probability density function.

Value

`dtopple` gives the density, `ptopple` gives the distribution function, `qtopple` gives the quantile function, and `rtopple` generates random deviates.

Note

The Topp-Leone distribution is related to the triangle distribution.

Author(s)

T. W. Yee

References

Topp, C. W. and F. C. Leone (1955). A family of J-shaped frequency functions. *Journal of the American Statistical Association*, **50**, 209–219.

See Also

topple, Triangle.

Examples

```
## Not run:  shape <- 0.7; x <- seq(0.02, 0.999, length = 300)
plot(x, dtopple(x, shape = shape), type = "l", col = "blue",
     main = "Blue is density, orange is CDF", ylab = "", las = 1,
     sub = "Purple lines are the 10,20,...,90 percentiles")
abline(h = 0, col = "blue", lty = 2)
lines(x, ptopple(x, shape = shape), type = "l", col = "orange")
probs <- seq(0.1, 0.9, by = 0.1)
Q <- qtopple(probs, shape = shape)
lines(Q, dtopple(Q, shape), col = "purple", lty = 3, type = "h")
lines(Q, ptopple(Q, shape), col = "purple", lty = 3, type = "h")
abline(h = probs, col = "purple", lty = 3)
max(abs(ptopple(Q, shape) - probs))  # Should be zero

## End(Not run)
```

---

topple *Topp-Leone Distribution Family Function*

Description

Estimating the parameter of the Topp-Leone distribution by maximum likelihood estimation.

Usage

```
topple(lshape = "logitlink", zero = NULL, gshape = ppoints(8),
       parallel = FALSE, percentiles = 50,
       type.fitted = c("mean", "percentiles", "Qlink"))
```

Arguments

- `lshape, gshape` Details at CommonVGAMffArguments. The CIA link is loglink, for shape approaching unity.
- `zero, parallel` Details at CommonVGAMffArguments.
- `type.fitted, percentiles` See CommonVGAMffArguments for information. Using "Qlink" is for quantile-links in **VGAMextra**.

Details

The Topple distribution has a probability density function that can be written

$$f(y; s) = 2s(1 - y)[y(2 - y)]^{s-1}$$

for $0 < y < 1$ and shape parameter $0 < s < 1$. The mean of Y is $1 - 4^s[\Gamma(1 + s)]^2/\Gamma(2 + 2s)$ (returned as the fitted values).

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Note

Fisher-scoring and Newton-Raphson are the same here. A related distribution is the triangle distribution. This **VGAM** family function handles multiple responses.

Author(s)

T. W. Yee

References

Topp, C. W. and F. C. Leone (1955). A family of J-shaped frequency functions. *Journal of the American Statistical Association*, **50**, 209–219.

See Also

Topple, Triangle.

Examples

```
tdata <- data.frame(y = rtopple(1000, logitlink(1, inverse = TRUE)))
tfit <- vglm(y ~ 1, topple, tdata, trace = TRUE, crit = "coef")
coef(tfit, matrix = TRUE)
Coef(tfit)
```

---

toxop *Toxoplasmosis Data*

Description

Toxoplasmosis data in 34 cities in El Salvador.

Usage

```
data(toxop)
```

Format

A data frame with 34 observations on the following 4 variables.

`rainfall` a numeric vector; the amount of rainfall in each city.

`ssize` a numeric vector; sample size.

`cityNo` a numeric vector; the city number.

`positive` a numeric vector; the number of subjects testing positive for the disease.

Details

See the references for details.

Source

See the references for details.

References

Efron, B. (1978). Regression and ANOVA With zero-one data: measures of residual variation. *Journal of the American Statistical Association*, **73**, 113–121.

Efron, B. (1986). Double exponential families and their use in generalized linear regression. *Journal of the American Statistical Association*, **81**, 709–721.

See Also

`double.expbinomial`.

Examples

```
## Not run:  with(toxop, plot(rainfall, positive/ssize, col = "blue"))
plot(toxop, col = "blue")
## End(Not run)
```

Triangle   *The Triangle Distribution*

Description

Density, distribution function, quantile function and random generation for the Triangle distribution with parameter `theta`.

Usage

```
dtriangle(x, theta, lower = 0, upper = 1, log = FALSE)
ptriangle(q, theta, lower = 0, upper = 1, lower.tail = TRUE, log.p = FALSE)
qtriangle(p, theta, lower = 0, upper = 1, lower.tail = TRUE, log.p = FALSE)
rtriangle(n, theta, lower = 0, upper = 1)
```

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
n	number of observations. Same as `runif`.
theta	the theta parameter which lies between `lower` and `upper`.
lower, upper	lower and upper limits of the distribution. Must be finite.
log	Logical. If `log = TRUE` then the logarithm of the density is returned.
lower.tail, log.p	Same meaning as in `pnorm` or `qnorm`.

Details

See `triangle`, the **VGAM** family function for estimating the parameter θ by maximum likelihood estimation.

Value

`dtriangle` gives the density, `ptriangle` gives the distribution function, `qtriangle` gives the quantile function, and `rtriangle` generates random deviates.

Author(s)

T. W. Yee and Kai Huang

See Also

`triangle`, `topple`.

Examples

```
## Not run:  x <- seq(-0.1, 1.1, by = 0.01); theta <- 0.75
plot(x, dtriangle(x, theta = theta), type = "l", col = "blue", las = 1,
     main = "Blue is density, orange is the CDF",
     sub = "Purple lines are the 10,20,...,90 percentiles",
     ylim = c(0,2), ylab = "")
abline(h = 0, col = "blue", lty = 2)
lines(x, ptriangle(x, theta = theta), col = "orange")
probs <- seq(0.1, 0.9, by = 0.1)
Q <- qtriangle(probs, theta = theta)
lines(Q, dtriangle(Q, theta = theta), col = "purple", lty = 3, type = "h")
ptriangle(Q, theta = theta) - probs  # Should be all zero
abline(h = probs, col = "purple", lty = 3)
## End(Not run)
```

`triangle` *Triangle Distribution Family Function*

Description

Estimating the parameter of the triangle distribution by maximum likelihood estimation.

Usage

```
triangle(lower = 0, upper = 1,
         link = extlogitlink(min = 0, max = 1), itheta = NULL)
```

Arguments

`lower, upper`	lower and upper limits of the distribution. Must be finite. Called A and B respectively below.
`link`	Parameter link function applied to the parameter θ, which lies in (A, B). See `Links` for more choices. The default constrains the estimate to lie in the interval.
`itheta`	Optional initial value for the parameter. The default is to compute the value internally.

Details

The triangle distribution has a probability density function that consists of two lines joined at θ, which is the location of the mode. The lines intersect the $y = 0$ axis at A and B. Here, Fisher scoring is used.

On fitting, the `extra` slot has components called `lower` and `upper` which contains the values of the above arguments (recycled to the right length). The fitted values are the mean of the distribution, which is $(A + B + \theta)/3$.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Warning

The MLE regularity conditions do not hold for this distribution (e.g., the first derivative evaluated at the mode does not exist because it is not continuous) so that misleading inferences may result, e.g., in the `summary` and `vcov` of the object. Additionally, convergence to the MLE often appears to fail.

Note

The response must contain values in (A, B). For most data sets (especially small ones) it is very common for half-stepping to occur.

Arguments `lower` and `upper` and `link` must match. For example, setting `lower = 0.2` and `upper = 4` and `link = extlogitlink(min = 0.2, max = 4.1)` will result in an error. Ideally `link = extlogitlink(min = lower, max = upper)` ought to work but it does not (yet)! Minimal error checking is done for this deficiency.

Author(s)

T. W. Yee

References

Kotz, S. and van Dorp, J. R. (2004). Beyond Beta: Other Continuous Families of Distributions with Bounded Support and Applications. Chapter 1. World Scientific: Singapore.

Nguyen, H. D. and McLachlan, G. J. (2016). Maximum likelihood estimation of triangular and polygon distributions. *Computational Statistics and Data Analysis*, **102**, 23–36.

See Also

`Triangle`, `Topple`, `simulate.vlm`.

Examples

```
## Not run:
# Example 1
tdata <- data.frame(y = rtriangle(n <- 3000, theta = 3/4))
fit <- vglm(y ~ 1, triangle(link = "identitylink"), tdata,
             trace = TRUE)
coef(fit, matrix = TRUE)
Coef(fit)
head(fit@extra$lower)
head(fitted(fit))
with(tdata, mean(y))

# Example 2; Kotz and van Dorp (2004), p.14
rdata <- data.frame(y = c(0.1,0.25,0.3,0.4,0.45, 0.6, 0.75, 0.8))
fit <- vglm(y ~ 1, triangle(link = "identitylink"), rdata,
            trace = TRUE, crit = "coef", maxit = 1000)
Coef(fit)  # The MLE is the 3rd order statistic, which is 0.3.
fit <- vglm(y ~ 1, triangle(link = "identitylink"), rdata,
            trace = TRUE, crit = "coef", maxit = 1001)
Coef(fit)  # The MLE is the 3rd order statistic, which is 0.3.

## End(Not run)
```

`trim.constraints` *Trimmed Constraint Matrices*

Description

Deletes statistically nonsignficant regression coefficients via their constraint matrices, for future refitting.

Usage

```
trim.constraints(object, sig.level = 0.05, max.num = Inf,
                 intercepts = TRUE, ...)
```

Arguments

- `object` Some **VGAM** object, especially having class `vglmff-class`. It has not yet been tested on non-"`vglm`" objects.
- `sig.level` Significance levels, with values in $[0, 1]$. Columns of constraint matices whose p-values are larger than this argument are deleted. With terms that generate more than one column of the "`lm`" model matrix, all p-values must be greater than this argument for deletion. This argument is recycled to the total number of regression coefficients of `object`.
- `max.num` Numeric, positive and integer-valued. Maximum number of regression coefficients allowable for deletion. This allows one to limit the number of deleted coefficients. For example, if `max.num = 1` then only the largest p-value is used for the deletion, provided it is larger than `sig.level`. The default is to delete all those coefficients whose p-values are greater than `sig.level`. With a finite value, this argument will probably not work properly when there are terms that generate more than one column of the LM model matrix. Having a value greater than unity might be unsuitable in the presence of multicollinearity because all correlated variables might be eliminated at once.
- `intercepts` Logical. Trim the intercept term? If `FALSE` then the constraint matrix for the "`(Intercept)`" term is left unchanged.
- `...` Unused but for provision in the future.

Details

This utility function is intended to simplify an existing `vglm` object having variables (terms) that affect unnecessary parameters. Suppose the explanatory variables in the formula includes a simple numeric covariate called `x2`. This variable will affect every linear predictor if `zero = NULL` in the **VGAM** family function. This situation may correspond to the constraint matrices having unnecessary columns because their regression coefficients are statistically nonsignificant. This function attempts to delete those columns and return a possibly simplified list of constraint matrices that can make refitting a simpler model easy to do. P-values obtained from `summaryvglm` (with `HDEtest = FALSE` for increased speed) are compared to `sig.level` to test for statistical significance.

For terms that generate more than one column of the "lm" model matrix, such as bs and poly, the column is deleted if all regression coefficients are statistically nonsignificant. Incidentally, users should instead use sm.bs, sm.ns, sm.poly, etc., for smart and safe prediction.

One can think of this function as facilitating *backward elimination* for variable selection, especially if max.num = 1 and $M = 1$, however usually more than one regression coefficient is deleted here by default.

Value

A list of possibly simpler constraint matrices that can be fed back into the model using the constraints argument (usually zero = NULL is needed to avoid a warning). Consequently, they are required to be of the "term"-type. After the model is refitted, applying summaryvglm should result in regression coefficients that are 'all' statistically significant.

Warning

This function has not been tested thoroughly. One extreme is that a term is totally deleted because none of its regression coefficients are needed, and that situation has not yet been finalized. Ideally, object only contains terms where at least one regression coefficient has a p-value less than sig.level. For ordered factors and other situations, deleting certain columns may not make sense and destroy interpretability.

As stated above, max.num may not work properly when there are terms that generate more than one column of the LM model matrix. However, this limitation may change in the future.

Note

This function is experimental and may be replaced by some other function in the future. This function does not use S4 object oriented programming but may be converted to such in the future.

Author(s)

T. W. Yee

See Also

constraints, vglm, summaryvglm, model.matrixvlm, drop1.vglm, step4vglm, sm.bs, sm.ns, sm.poly.

Examples

```
## Not run:  data("xs.nz", package = "VGAMdata")
fit1 <-
  vglm(cbind(worry, worrier) ~ bs(age) + sex + ethnicity + cat + dog,
       binom2.or(zero = NULL), data = xs.nz, trace = TRUE)
summary(fit1, HDEtest = FALSE)  # 'cat' is not significant at all
dim(constraints(fit1, matrix = TRUE))
(tclist1 <- trim.constraints(fit1))  # No 'cat'
fit2 <-  # Delete 'cat' manually from the formula:
  vglm(cbind(worry, worrier) ~ bs(age) + sex + ethnicity +       dog,
       binom2.or(zero = NULL), data = xs.nz,
```

```
        constraints = tclist1, trace = TRUE)
summary(fit2, HDEtest = FALSE)  # A simplified model
dim(constraints(fit2, matrix = TRUE))  # Fewer regression coefficients

## End(Not run)
```

Trinorm *Trivariate Normal Distribution Density and Random Variates*

Description

Density and random generation for the trivariate normal distribution distribution.

Usage

```
dtrinorm(x1, x2, x3, mean1 = 0, mean2 = 0, mean3 = 0,
         var1 = 1, var2 = 1, var3 = 1,
         cov12 = 0, cov23 = 0, cov13 = 0, log = FALSE)
rtrinorm(n,          mean1 = 0, mean2 = 0, mean3 = 0,
         var1 = 1, var2 = 1, var3 = 1,
         cov12 = 0, cov23 = 0, cov13 = 0)
```

Arguments

x1, x2, x3	vector of quantiles.
mean1, mean2, mean3	vectors of means.
var1, var2, var3	vectors of variances.
cov12, cov23, cov13	vectors of covariances.
n	number of observations. Same as rnorm.
log	Logical. If log = TRUE then the logarithm of the density is returned.

Details

The default arguments correspond to the standard trivariate normal distribution with correlation parameters equal to 0, which corresponds to three independent standard normal distributions. Let sd1 (say) be sqrt(var1) and written σ_1, etc. Then the general formula for each correlation coefficient is of the form $\rho_{12} = cov_{12}/(\sigma_1\sigma_2)$, and similarly for the two others. Thus if the var arguments are left alone then the cov can be inputted with ρs.

Value

dtrinorm gives the density, rtrinorm generates random deviates (n by 3 matrix).

Warning

dtrinorm()'s arguments might change in the future! It's safest to use the full argument names to future-proof possible changes!

Note

For rtrinorm(), if the ith variance-covariance matrix is not positive-definite then the ith row is all NAs.

See Also

pnorm, trinormal, uninormal, binormal, rbinorm.

Examples

```
## Not run: nn <- 1000
tdata <- data.frame(x2 = sort(runif(nn)))
tdata <- transform(tdata, mean1 = 1 + 2 * x2,
                   mean2 = 3 + 1 * x2, mean3 = 4,
                   var1 = exp( 1), var2 = exp( 1), var3 = exp( 1),
                   rho12 = rhobitlink( 1, inverse = TRUE),
                   rho23 = rhobitlink( 1, inverse = TRUE),
                   rho13 = rhobitlink(-1, inverse = TRUE))
ymat <- with(tdata, rtrinorm(nn, mean1, mean2, mean3,
                             var1, var2, var3,
                             sqrt(var1)*sqrt(var1)*rho12,
                             sqrt(var2)*sqrt(var3)*rho23,
                             sqrt(var1)*sqrt(var3)*rho13))
pairs(ymat, col = "blue")

## End(Not run)
```

---

trinormal *Trivariate Normal Distribution Family Function*

Description

Maximum likelihood estimation of the nine parameters of a trivariate normal distribution.

Usage

```
trinormal(zero = c("sd", "rho"), eq.mean = FALSE,
  eq.sd = FALSE, eq.cor = FALSE,
  lmean1 = "identitylink", lmean2 = "identitylink",
  lmean3 = "identitylink",
  lsd1   = "loglink", lsd2   = "loglink", lsd3   = "loglink",
  lrho12 = "rhobitlink", lrho23 = "rhobitlink", lrho13 = "rhobitlink",
  imean1 = NULL, imean2 = NULL, imean3 = NULL,
  isd1   = NULL, isd2   = NULL, isd3   = NULL,
  irho12 = NULL, irho23 = NULL, irho13 = NULL, imethod = 1)
```

Arguments

`lmean1, lmean2, lmean3, lsd1, lsd2, lsd3`
: Link functions applied to the means and standard deviations. See `Links` for more choices. Being positive quantities, a log link is the default for the standard deviations.

`lrho12, lrho23, lrho13`
: Link functions applied to the correlation parameters. See `Links` for more choices. By default the correlation parameters are allowed to have a value between -1 and 1, but that may be problematic when `eq.cor = TRUE` because they should have a value between -0.5 and 1.

`imean1, imean2, imean3, isd1, isd2, isd3`
: See `CommonVGAMffArguments` for more information.

`irho12, irho23, irho13, imethod, zero`
: See `CommonVGAMffArguments` for more information.

`eq.mean, eq.sd, eq.cor`
: Logical. Constrain the means or the standard deviations or correlation parameters to be equal?

Details

For the trivariate normal distribution, this fits a linear model (LM) to the means, and by default, the other parameters are intercept-only. The response should be a three-column matrix. The three correlation parameters are prefixed by `rho`, and the default gives them values between -1 and 1 however, this may be problematic when the correlation parameters are constrained to be equal, etc.. The fitted means are returned as the fitted values, which is in the form of a three-column matrix. Fisher scoring is implemented.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Warning

The default parameterization does not make the estimated variance-covariance matrix positive-definite. In order for the variance-covariance matrix to be positive-definite the quantity `1 - rho12^2 - rho13^2 - rho23^2 + 2 * rho12 * rho13 * rho23` must be positive, and if `eq.cor = TRUE` then this means that the `rho`s must be between -0.5 and 1.

Author(s)

T. W. Yee

See Also

`uninormal`, `binormal`, `rtrinorm`.

Examples

```
## Not run:   set.seed(123); nn <- 1000
tdata <- data.frame(x2 = runif(nn), x3 = runif(nn))
tdata <- transform(tdata, y1 = rnorm(nn, 1 + 2 * x2),
                          y2 = rnorm(nn, 3 + 4 * x2),
                          y3 = rnorm(nn, 4 + 5 * x2))
fit1 <- vglm(cbind(y1, y2, y3) ~ x2, data = tdata,
             trinormal(eq.sd = TRUE, eq.cor = TRUE), trace = TRUE)
coef(fit1, matrix = TRUE)
constraints(fit1)
summary(fit1)
# Try this when eq.sd = TRUE, eq.cor = TRUE:
fit2 <-
  vglm(cbind(y1, y2, y3) ~ x2, data = tdata, stepsize = 0.25,
       trinormal(eq.sd = TRUE, eq.cor = TRUE,
                 lrho12 = extlogitlink(min = -0.5),
                 lrho23 = extlogitlink(min = -0.5),
                 lrho13 = extlogitlink(min = -0.5)), trace = TRUE)
coef(fit2, matrix = TRUE)

## End(Not run)
```

---

trplot *Trajectory Plot*

Description

Generic function for a trajectory plot.

Usage

```
trplot(object, ...)
```

Arguments

`object`	An object for which a trajectory plot is meaningful.
`...`	Other arguments fed into the specific methods function of the model. They usually are graphical parameters, and sometimes they are fed into the methods function for `Coef`.

Details

Trajectory plots can be defined in different ways for different models. Many models have no such notion or definition.

For quadratic and additive ordination models they plot the fitted values of two species against each other (more than two is theoretically possible, but not implemented in this software yet).

Value

The value returned depends specifically on the methods function invoked.

Author(s)

Thomas W. Yee

References

Yee, T. W. (2020). On constrained and unconstrained quadratic ordination. *Manuscript in preparation*.

See Also

`trplot.qrrvglm`, `perspqrrvglm`, `lvplot`.

Examples

```
## Not run:  set.seed(123)
hspider[, 1:6] <- scale(hspider[, 1:6])  # Stdze environ. vars
p1cqo <- cqo(cbind(Alopacce, Alopcune, Alopfabr, Arctlute,
                   Arctperi, Auloalbi, Pardlugu, Pardmont,
                   Pardnigr, Pardpull, Trocterr, Zoraspin) ~
            WaterCon + BareSand + FallTwig +
            CoveMoss + CoveHerb + ReflLux,
            poissonff, data = hspider, Crow1positive = FALSE)

nos <- ncol(depvar(p1cqo))
clr <- 1:nos  # OR (1:(nos+1))[-7]  to omit yellow

trplot(p1cqo, which.species = 1:3, log = "xy", lwd = 2,
       col = c("blue", "orange", "green"), label = TRUE) -> ii
legend(0.00005, 0.3, paste(ii$species[, 1], ii$species[, 2],
                           sep = " and "),
       lwd = 2, lty = 1, col = c("blue", "orange", "green"))
abline(a = 0, b = 1, lty = "dashed", col = "grey")
## End(Not run)
```

`trplot.qrrvglm` *Trajectory plot for QRR-VGLMs*

Description

Produces a trajectory plot for *quadratic reduced-rank vector generalized linear models* (QRR-VGLMs). It is only applicable for rank-1 models with argument noRRR = ~ 1.

Usage

```
trplot.qrrvglm(object, which.species = NULL, add = FALSE,
               show.plot = TRUE,
               label.sites = FALSE, sitenames = rownames(object@y),
               axes.equal = TRUE, cex = par()$cex,
               col = 1:(nos * (nos - 1)/2), log = "",
               lty = rep_len(par()$lty, nos * (nos - 1)/2),
               lwd = rep_len(par()$lwd, nos * (nos - 1)/2),
               tcol = rep_len(par()$col, nos * (nos - 1)/2),
               xlab = NULL, ylab = NULL,
               main = "", type = "b", check.ok = TRUE, ...)
```

Arguments

`object`	Object of class "qrrvglm", i.e., a CQO object.
`which.species`	Integer or character vector specifying the species to be plotted. If integer, these are the columns of the response matrix. If character, these must match exactly with the species' names. The default is to use all species.
`add`	Logical. Add to an existing plot? If FALSE (default), a new plot is made.
`show.plot`	Logical. Plot it?
`label.sites`	Logical. If TRUE, the points on the curves/trajectories are labelled with the `sitenames`.
`sitenames`	Character vector. The names of the sites.
`axes.equal`	Logical. If TRUE, the x- and y-axes will be on the same scale.
`cex`	Character expansion of the labelling of the site names. Used only if `label.sites` is TRUE. See the `cex` argument in `par`.
`col`	Color of the lines. See the `col` argument in `par`. Here, `nos` is the number of species.
`log`	Character, specifying which (if any) of the x- and y-axes are to be on a logarithmic scale. See the `log` argument in `par`.
`lty`	Line type. See the `lty` argument of `par`.
`lwd`	Line width. See the `lwd` argument of `par`.
`tcol`	Color of the text for the site names. See the `col` argument in `par`. Used only if `label.sites` is TRUE.
`xlab`	Character caption for the x-axis. By default, a suitable caption is found. See the `xlab` argument in `plot` or `title`.
`ylab`	Character caption for the y-axis. By default, a suitable caption is found. See the `xlab` argument in `plot` or `title`.
`main`	Character, giving the title of the plot. See the `main` argument in `plot` or `title`.
`type`	Character, giving the type of plot. A common option is to use `type="l"` for lines only. See the `type` argument of `plot`.
`check.ok`	Logical. Whether a check is performed to see that noRRR = ~ 1 was used. It doesn't make sense to have a trace plot unless this is so.
`...`	Arguments passed into the `plot` function when setting up the entire plot. Useful arguments here include `xlim` and `ylim`.

Details

A trajectory plot plots the fitted values of a 'second' species against a 'first' species. The argument `which.species` must therefore contain at least two species. By default, all of the species that were fitted in `object` are plotted. With more than a few species the resulting plot will be very congested, and so it is recommended that only a few species be selected for plotting.

In the above, M is the number of species selected for plotting, so there will be $M(M-1)/2$ curves/trajectories in total.

A trajectory plot will be fitted only if `noRRR = ~ 1` because otherwise the trajectory will not be a smooth function of the latent variables.

Value

A list with the following components.

`species.names` A matrix of characters giving the 'first' and 'second' species. The number of different combinations of species is given by the number of rows. This is useful for creating a legend.

`sitenames` A character vector of site names, sorted by the latent variable (from low to high).

Note

Plotting the axes on a log scale is often a good idea. The use of `xlim` and `ylim` to control the axis limits is also a good idea, so as to limit the extent of the curves at low abundances or probabilities. Setting `label.sites = TRUE` is a good idea only if the number of sites is small, otherwise there is too much clutter.

Author(s)

Thomas W. Yee

References

Yee, T. W. (2020). On constrained and unconstrained quadratic ordination. *Manuscript in preparation*.

See Also

`cqo`, `par`, `title`.

Examples

```
## Not run:  set.seed(111)  # Leads to the global solution
# hspider[,1:6] <- scale(hspider[,1:6])  # Stdze the environ vars
p1 <- cqo(cbind(Alopacce, Alopcune, Alopfabr, Arctlute,
                Arctperi, Auloalbi, Pardlugu, Pardmont,
                Pardnigr, Pardpull, Trocterr, Zoraspin) ~
          WaterCon + BareSand + FallTwig + CoveMoss +
          CoveHerb + ReflLux,
          poissonff, data = hspider, trace = FALSE)
```

```
trplot(p1, which.species = 1:3, log = "xy", type = "b", lty = 1,
       main = "Trajectory plot of three hunting spiders species",
       col = c("blue","red","green"), lwd = 2, label = TRUE) -> ii
legend(0.00005, 0.3, lwd = 2, lty = 1,
       col = c("blue", "red", "green"),
       with(ii, paste(species.names[,1], species.names[,2],
                      sep = " and ")))
abline(a = 0, b = 1, lty = "dashed", col = "grey")  # Ref. line

## End(Not run)
```

`Trunc` *Truncated Values for the GT-Expansion Method*

Description

Given the minimum and maximum values in a response variable, and a positive multiplier, returns the truncated values for generally-truncated regression

Usage

```
Trunc(Range, mux = 2, location = 0, omits = TRUE)
```

Arguments

`Range`	Numeric, of length 2 containing the minimum and maximum (in that order) of the untransformed data. Alternatively, if `length(Range) > 2` then it is assumed that the entire untransformed data is passed in so that `range` is applied.
`mux`	Numeric, the multiplier. A positive integer.
`location`	Numeric, the location parameter, allows a shift to the right.
`omits`	Logical. The default is to return the truncated values (those being omitted). If `FALSE` then the multiples are returned.

Details

Generally-truncated regression can handle underdispersion with respect to some parent or base distribution such as the Poisson. Yee and Ma (2023) call this the *GT-Expansion* (GTE) method, which is a special case of the GT-location-scale (GT-LS) method. This is a utility function to help make life easier. It is assumed that the response is a count variable.

Value

A vector of values to be fed into the `truncate` argument of a **VGAM** family function such as `gaitdpoisson`. If `mux = 1` then the function will return a `NULL` rather than `integer(0)`.

Author(s)

T. W. Yee

See Also

gaitdpoisson, gaitdlog, gaitdzeta, range, setdiff, goffset.

Examples

```
Trunc(c(1, 8), 2)

## Not run:
set.seed(1)  # The following example is based on the normal
mymean <- 20; m.truth <- 3  # approximation to the Poisson.
gdata <- data.frame(y1 = round(rnorm((nn <- 1000), mymean,
                                     sd = sqrt(mymean / m.truth))))
org1 <- with(gdata, range(y1))  # Original range of the raw data
m.max <- 5  # Try multipliers 1:m.max
logliks <- numeric(m.max)
names(logliks) <- as.character(1:m.max)
for (i in 1:m.max) {
  logliks[i] <- logLik(vglm(i * y1 ~ offset(rep(log(i), nn)),
    gaitdpoisson(truncate = Trunc(org1, i)), data = gdata))
}
sort(logliks, decreasing = TRUE)  # Best to worst
 par(mfrow = c(1, 2))
plot(with(gdata, table(y1)))  # Underdispersed wrt Poisson
plot(logliks, col = "blue", type = "b", xlab = "Multiplier")
## End(Not run)
```

Truncpareto *The Truncated Pareto Distribution*

Description

Density, distribution function, quantile function and random generation for the upper truncated Pareto(I) distribution with parameters lower, upper and shape.

Usage

```
dtruncpareto(x, lower, upper, shape, log = FALSE)
ptruncpareto(q, lower, upper, shape, lower.tail = TRUE, log.p = FALSE)
qtruncpareto(p, lower, upper, shape)
rtruncpareto(n, lower, upper, shape)
```

Arguments

`x, q` vector of quantiles.

`p` vector of probabilities.

`n, log` Same meaning as `runif`.

`lower, upper, shape`
the lower, upper and shape (k) parameters. If necessary, values are recycled.

`lower.tail, log.p`
Same meaning as in `pnorm` or `qnorm`.

Details

See `truncpareto`, the **VGAM** family function for estimating the parameter k by maximum likelihood estimation, for the formula of the probability density function and the range restrictions imposed on the parameters.

Value

`dtruncpareto` gives the density, `ptruncpareto` gives the distribution function, `qtruncpareto` gives the quantile function, and `rtruncpareto` generates random deviates.

Author(s)

T. W. Yee and Kai Huang

References

Aban, I. B., Meerschaert, M. M. and Panorska, A. K. (2006). Parameter estimation for the truncated Pareto distribution, *Journal of the American Statistical Association*, **101**(473), 270–277.

See Also

`truncpareto`.

Examples

```
 lower <- 3; upper <- 8; kay <- exp(0.5)
## Not run:  xx <- seq(lower - 0.5, upper + 0.5, len = 401)
plot(xx, dtruncpareto(xx, low = lower, upp = upper, shape = kay),
     main = "Truncated Pareto density split into 10 equal areas",
     type = "l", ylim = 0:1, xlab = "x")
abline(h = 0, col = "blue", lty = 2)
qq <- qtruncpareto(seq(0.1, 0.9, by = 0.1), low = lower, upp = upper,
                   shape = kay)
lines(qq, dtruncpareto(qq, low = lower, upp = upper, shape = kay),
      col = "purple", lty = 3, type = "h")
lines(xx, ptruncpareto(xx, low = lower, upp = upper, shape = kay),
      col = "orange")
## End(Not run)
pp <- seq(0.1, 0.9, by = 0.1)
```

```
qq <- qtruncpareto(pp, lower = lower, upper = upper, shape = kay)

ptruncpareto(qq, lower = lower, upper = upper, shape = kay)
qtruncpareto(ptruncpareto(qq, lower = lower, upper = upper, shape = kay),
         lower = lower, upper = upper, shape = kay) - qq  # Should be all 0
```

---

truncweibull *Truncated Weibull Distribution Family Function*

Description

Maximum likelihood estimation of the 2-parameter Weibull distribution with lower truncation. No observations should be censored.

Usage

```
truncweibull(lower.limit = 1e-5,
             lAlpha = "loglink", lBetaa = "loglink",
             iAlpha = NULL,   iBetaa = NULL,
             nrfs = 1, probs.y = c(0.2, 0.5, 0.8),
             imethod = 1, zero = "Betaa")
```

Arguments

`lower.limit` Positive lower truncation limits. Recycled to the same dimension as the response, going across rows first. The default, being close to 0, should mean effectively the same results as `weibullR` if there are no response values that are smaller.

`lAlpha, lBetaa` Parameter link functions applied to the (positive) parameters `Alpha` (called α below) and (positive) `Betaa` (called β below). See `Links` for more choices.

`iAlpha, iBetaa` See `CommonVGAMffArguments`.

`imethod, nrfs, zero, probs.y`
Details at `weibullR` and `CommonVGAMffArguments`.

Details

MLE of the two parameters of the Weibull distribution are computed, subject to lower truncation. That is, all response values are greater than `lower.limit`, element-wise. For a particular observation this is any known positive value. This function is currently based directly on Wingo (1989) and his parameterization is used (it differs from `weibullR`.) In particular, $\beta = a$ and $\alpha = (1/b)^a$ where a and b are as in `weibullR` and `dweibull`.

Upon fitting the `extra` slot has a component called `lower.limit` which is of the same dimension as the response. The fitted values are the mean, which are computed using `pgamma.deriv` and `pgamma.deriv.unscaled`.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Warning

This function may be converted to the same parameterization as weibullR at any time. Yet to do: one element of the EIM may be wrong (due to two interpretations of a formula; but it seems to work). Convergence is slower than usual and this may imply something is wrong; use argument maxit. In fact, it's probably because pgamma.deriv.unscaled is inaccurate at q = 1 and q = 2. Also, convergence should be monitored, especially if the truncation means that a large proportion of the data is lost compared to an ordinary Weibull distribution.

Note

More improvements need to be made, e.g., initial values are currently based on no truncation. This **VGAM** family function handles multiple responses.

Author(s)

T. W. Yee

References

Wingo, D. R. (1989). The left-truncated Weibull distribution: theory and computation. *Statistical Papers*, **30**(1), 39–48.

See Also

weibullR, dweibull, pgamma.deriv, pgamma.deriv.unscaled.

Examples

```
nn <- 5000; prop.lost <- 0.40   # Proportion lost to truncation
wdata <- data.frame(x2 = runif(nn))  # Complete Weibull data
wdata <- transform(wdata,
         Betaa = exp(1))  # > 2 okay (satisfies regularity conds)
wdata <- transform(wdata, Alpha = exp(0.5 - 1 * x2))
wdata <- transform(wdata, Shape = Betaa,
#                         aaa   = Betaa,
#                         bbb   = 1 / Alpha^(1 / Betaa),
                          Scale = 1 / Alpha^(1 / Betaa))
wdata <- transform(wdata, y2 = rweibull(nn, Shape, scale = Scale))
summary(wdata)

# Proportion lost:
lower.limit2 <- with(wdata, quantile(y2, prob = prop.lost))
# Smaller due to truncation:
wdata <- subset(wdata, y2 > lower.limit2)

fit1 <- vglm(y2 ~ x2, maxit = 100, trace = TRUE,
```

```
             truncweibull(lower.limit = lower.limit2), wdata)
coef(fit1, matrix = TRUE)
summary(fit1)
vcov(fit1)
head(fit1@extra$lower.limit)
```

---

ucberk *University California Berkeley Graduate Admissions*

Description

University California Berkeley Graduate Admissions: counts cross-classified by acceptance/rejection and gender, for the six largest departments.

Usage

```
data(ucberk)
```

Format

A data frame with 6 departmental groups with the following 5 columns.

m.deny Counts of men denied admission.

m.admit Counts of men admitted.

w.deny Counts of women denied admission.

w.admit Counts of women admitted.

dept Department (the six largest), called `A`, `B`, ..., `F`.

Details

From Bickel et al. (1975), the data consists of applications for admission to graduate study at the University of California, Berkeley, for the fall 1973 quarter. In the admissions cycle for that quarter, the Graduate Division at Berkeley received approximately 15,000 applications, some of which were later withdrawn or transferred to a different proposed entry quarter by the applicants. Of the applications finally remaining for the fall 1973 cycle 12,763 were sufficiently complete to permit a decision. There were about 101 graduate department and interdepartmental graduate majors. There were 8442 male applicants and 4321 female applicants. About 44 percent of the males and about 35 percent of the females were admitted. The data are well-known for illustrating Simpson's paradox.

References

Bickel, P. J., Hammel, E. A. and O'Connell, J. W. (1975). Sex bias in graduate admissions: data from Berkeley. *Science*, **187**(4175): 398–404.

Freedman, D., Pisani, R. and Purves, R. (1998). Chapter 2 of *Statistics*, 3rd. ed., W. W. Norton & Company.

Examples

```
summary(ucberk)
```

---

uninormal *Univariate Normal Distribution*

Description

Maximum likelihood estimation of the two parameters of a univariate normal distribution.

Usage

```
uninormal(lmean = "identitylink", lsd = "loglink", lvar =
          "loglink", var.arg = FALSE, imethod = 1, isd = NULL,
          parallel = FALSE, vfl = FALSE, Form2 = NULL,
 smallno = 1e-05, zero = if (var.arg)
          "var" else "sd")
```

Arguments

`lmean, lsd, lvar`
: Link functions applied to the mean and standard deviation/variance. See `Links` for more choices. Being positive quantities, a log link is the default for the standard deviation and variance (see `var.arg`).

`var.arg`
: Logical. If `TRUE` then the second parameter is the variance and `lsd` and `esd` are ignored, else the standard deviation is used and `lvar` and `evar` are ignored.

`smallno`
: Numeric, positive but close to 0. Used specifically for quasi-variances; if the link for the mean is `explink` then any non-positive value of `eta` is replaced by this quantity (hopefully, temporarily and only during early iterations).

`imethod, parallel, isd, zero`
: See `CommonVGAMffArguments` for information. If `lmean = loglink` then try `imethod = 2`. If `parallel = TRUE` then the parallelism constraint is not applied to the intercept.

`vfl, Form2`
: See `CommonVGAMffArguments` for information.

Details

This fits a linear model (LM) as the first linear/additive predictor. So, by default, this is just the mean. By default, the log of the standard deviation is the second linear/additive predictor. The Fisher information matrix is diagonal. This **VGAM** family function can handle multiple responses.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Warning

gaussianff() was deprecated but has been brought back into **VGAM** nominally. It should be called Mickey Mouse. It gives a warning and calls uninormal instead (hopefully all the arguments should pass in correctly). Users should avoid calling gaussianff(); use glm with gaussian instead. It is dangerous to treat what is an uninormal fit as a gaussianff() object.

Note

Yet to do: allow an argument such as eq.sd that enables the standard devations to be the same. Also, this function used to be called normal1() too, but it has been decommissioned.

Author(s)

T. W. Yee

References

Forbes, C., Evans, M., Hastings, N. and Peacock, B. (2011). *Statistical Distributions*, Hoboken, NJ, USA: John Wiley and Sons, Fourth edition.

See Also

posnormal, mix2normal, ordsup, normal.vcm, Qvar, tobit, cens.normal, foldnormal, skewnormal, double.cens.normal, SURff, AR1, huber2, studentt, binormal, trinormal, dnorm, simulate.vlm, hdeff.vglm.

Examples

```
udata <- data.frame(x2 = rnorm(nn <- 200))
udata <- transform(udata,
           y1  = rnorm(nn, m = 1 - 3*x2, sd = exp(1 + 0.2*x2)),
           y2a = rnorm(nn, m = 1 + 2*x2, sd = exp(1 + 2.0*x2)^0.5),
           y2b = rnorm(nn, m = 1 + 2*x2, sd = exp(1 + 2.0*x2)^0.5))
fit1 <- vglm(y1 ~ x2, uninormal(zero = NULL), udata, trace = TRUE)
coef(fit1, matrix = TRUE)
fit2 <- vglm(cbind(y2a, y2b) ~ x2, data = udata, trace = TRUE,
             uninormal(var = TRUE, parallel = TRUE ~ x2,
                       zero = NULL))
coef(fit2, matrix = TRUE)

# Generate data from N(mu=theta=10, sigma=theta) and estimate theta.
theta <- 10
udata <- data.frame(y3 = rnorm(100, m = theta, sd = theta))
fit3a <- vglm(y3 ~ 1, uninormal(lsd = "identitylink"), data = udata,
             constraints = list("(Intercept)" = rbind(1, 1)))
fit3b <- vglm(y3 ~ 1, uninormal(lsd = "identitylink",
                        parallel = TRUE ~ 1, zero = NULL), udata)
coef(fit3a, matrix = TRUE)
coef(fit3b, matrix = TRUE)  # Same as fit3a
```

UtilitiesVGAM *Utility Functions for the VGAM Package*

Description

A set of common utility functions used by **VGAM** family functions.

Usage

```
param.names(string, S = 1, skip1 = FALSE, sep = "")
dimm(M, hbw = M)
interleave.VGAM(.M, M1, inverse = FALSE)
```

Arguments

`string`	Character. Name of the parameter.
`M, .M`	Numeric. The total number of linear/additive predictors, called M. By total, it is meant summed over the number of responses. Often, M is the total number of parameters to be estimated (but this is not the same as the number of regression coefficients, unless the RHS of the formula is an intercept-only). The use of `.M` is unfortunate, but it is a compromise solution to what is presented in Yee (2015). Ideally, `.M` should be just `M`.
`M1`	Numeric. The number of linear/additive predictors for one response, called M_1. This argument used to be called `M`, but is now renamed properly.
`inverse`	Logical. Useful for the inverse function of `interleave.VGAM()`.
`S`	Numeric. The number of responses.
`skip1, sep`	The former is logical; should one skip (or omit) `"1"` when `S = 1`? The latter is the same argument as `paste`.
`hbw`	Numeric. The half-bandwidth, which measures the number of bands emanating from the central diagonal band.

Details

See Yee (2015) for some details about some of these functions.

Value

For `param.names()`, this function returns the parameter names for S responses, i.e., `string` is returned unchanged if $S = 1$, else `paste(string, 1:S, sep = "")`.

For `dimm()`, this function returns the number of elements to be stored for each of the working weight matrices. They are represented as columns in the matrix `wz` in e.g., `vglm.fit()`. See the *matrix-band* format described in Section 18.3.5 of Yee (2015).

For `interleave.VGAM()`, this function returns a reordering of the linear/additive predictors depending on the number of responses. The arguments presented in Table 18.5 may not be valid in your version of Yee (2015).

Author(s)

T. W. Yee. Victor Miranda added the `inverse` argument to `interleave.VGAM()`.

References

Yee, T. W. (2015). Vector Generalized Linear and Additive Models: With an Implementation in R. New York, USA: *Springer*.

See Also

CommonVGAMffArguments, VGAM-package.

Examples

```
param.names("shape", 1)  # "shape"
param.names("shape", 3)  # c("shape1", "shape2", "shape3")

dimm(3, hbw = 1)  # Diagonal matrix; the 3 elements need storage.
dimm(3)  # A general 3 x 3 symmetrix matrix has 6 unique elements.
dimm(3, hbw = 2)  # Tridiagonal matrix; the 3-3 element is 0 and unneeded.

M1 <- 2; ncoly <- 3; M <- ncoly * M1
mynames1 <- param.names("location", ncoly)
mynames2 <- param.names("scale",    ncoly)
(parameters.names <- c(mynames1, mynames2)[interleave.VGAM(M, M1 = M1)])
# The  following is/was in Yee (2015) and has a poor/deceptive style:
(parameters.names <- c(mynames1, mynames2)[interleave.VGAM(M, M  = M1)])
parameters.names[interleave.VGAM(M, M1 = M1, inverse = TRUE)]
```

V1 *V1 Flying-Bombs Hits in London*

Description

A small count data set. During WWII V1 flying-bombs were fired from sites in France (Pas-de-Calais) and Dutch coasts towards London. The number of hits per square grid around London were recorded.

Usage

```
data(V1)
```

Format

A data frame with the following variables.

hits Values between 0 and 4, and 7. Actually, the 7 is really imputed from the paper (it was recorded as "5 and over").

ofreq Observed frequency, i.e., the number of grids with that many hits.

Details

The data concerns 576 square grids each of 0.25 square kms about south London. The area was selected comprising 144 square kms over which the basic probability function of the distribution was very nearly constant. V1s, which were one type of flying-bomb, were a "Vergeltungswaffen" or vengeance weapon fired during the summer of 1944 at London. The V1s were informally called Buzz Bombs or Doodlebugs, and they were pulse-jet-powered with a warhead of 850 kg of explosives. Over 9500 were launched at London, and many were shot down by artillery and the RAF. Over the period considered the total number of bombs within the area was 537.

It was asserted that the bombs tended to be grouped in clusters. However, a basic Poisson analysis shows this is not the case. Their guidance system being rather primitive, the data is consistent with a Poisson distribution (random).

Compared to Clarke (1946), the more modern analysis of Shaw and Shaw (2019). shows a higher density of hits in south London, hence the distribution is not really uniform over the entire region.

Source

Clarke, R. D. (1946). An application of the Poisson distribution. *Journal of the Institute of Actuaries*, **72**(3), 481.

References

Shaw, L. P. and Shaw, L. F. (2019). The flying bomb and the actuary. *Significance*, **16**(5): 12–17.

See Also

V2, poissonff.

Examples

```
V1
mean(with(V1, rep(hits, times = ofreq)))
 var(with(V1, rep(hits, times = ofreq)))
 sum(with(V1, rep(hits, times = ofreq)))
## Not run:  barplot(with(V1, ofreq),
          names.arg = as.character(with(V1, hits)),
          main = "London V1 buzz bomb hits",
          col = "lightblue", las = 1,
          ylab = "Frequency", xlab = "Hits")
## End(Not run)
```

V2 — *V2 Missile Hits in London*

Description

A small count data set. During WWII V2 missiles were fired from the continent mainly towards London. The number of hits per square grid around London were recorded.

Usage

```
data(V2)
```

Format

A data frame with the following variables.

hits Values between 0 and 3.

ofreq Observed frequency, i.e., the number of grids with that many hits.

Details

The data concerns 408 square grids each of 0.25 square kms about south London (south of the River Thames). They were picked in a rectangular region of 102 square kilometres where the density of hits were roughly uniformly distributed. The data is somewhat comparable to `V1` albeit is a smaller data set.

Source

Shaw, L. P. and Shaw, L. F. (2019). The flying bomb and the actuary. *Significance*, **16**(5): 12–17.

See Also

`V1`, `poissonff`.

Examples

```
V2
mean(with(V2, rep(hits, times = ofreq)))
 var(with(V2, rep(hits, times = ofreq)))
 sum(with(V2, rep(hits, times = ofreq)))
## Not run:  barplot(with(V2, ofreq),
          names.arg = as.character(with(V2, hits)),
          main = "London V2 rocket hits",
          col = "lightgreen", las = 1,
          ylab = "Frequency", xlab = "Hits")
## End(Not run)
```

---

vcovvlm *Calculate Variance-Covariance Matrix for a Fitted VLM or RR-VGLM or DRR-VGLM or QRR-VGLM Object*

Description

Returns the variance-covariance matrix of the parameters of a fitted `vlm-class` or `rrvglm-class` or `drrvglm-class` object.

Usage

```
vcovvlm(object, dispersion = NULL, untransform = FALSE,
        complete = TRUE, ...)
vcovrrvglm(object, ...)
vcovdrrvglm(object, ...)
vcovqrrvglm(object, ...)
```

Arguments

`object`	A fitted model object, having class `vlm-class` or `rrvglm-class` or `drrvglm-class` or `qrrvglm-class` or a superclass of such. The former includes a `vglm` object.
`dispersion`	Numerical. This argument should not be used as **VGAM** will be phasing out dispersion parameters. Formerly, a value may be specified, else it is estimated for quasi-GLMs (e.g., method of moments). For almost all other types of VGLMs it is usually unity. The value is multiplied by the raw variance-covariance matrix.
`untransform`	logical. For intercept-only models with trivial constraints; if set `TRUE` then the parameter link function is inverted to give the answer for the untransformed/raw parameter.
`complete`	An argument currently ignored. Added only so that `linearHypothesis()` in **car** can be called.
`...`	Same as `vcov`.

Details

This methods function is based on the QR decomposition of the (large) VLM model matrix and working weight matrices. Currently `vcovvlm` operates on the fundamental `vlm-class` objects because pretty well all modelling functions in **VGAM** inherit from this. Currently `vcovrrvglm` is not entirely reliable because the elements of the **A**–**C** part of the matrix sometimes cannot be computed very accurately, so that the entire matrix is not positive-definite.

For `"qrrvglm"` objects, `vcovqrrvglm` is currently working with `Rank = 1` objects or when `I.tolerances = TRUE`. Then the answer is conditional given **C**. The code is based on `model.matrixqrrvglm` so that the `dimnames` are the same.

Value

Same as `vcov`.

Note

For some models inflated standard errors can occur, such as parameter estimates near the boundary of the parameter space. Detection for this is available for some models using `hdeff.vglm`, which tests for an Hauck-Donner effect (HDE) for each regression coefficient. If the HDE is present, using `lrt.stat.vlm` should return more accurate p-values.

Author(s)

Thomas W. Yee

See Also

confintvglm, summaryvglm, vcov, hdeff.vglm, lrt.stat.vlm, model.matrixqrrvglm.

Examples

```
## Not run:
ndata <- data.frame(x2 = runif(nn <- 300))
ndata <- transform(ndata, y1 = rnbinom(nn, mu = exp(3+x2), exp(1)),
                          y2 = rnbinom(nn, mu = exp(2-x2), exp(0)))
fit1 <- vglm(cbind(y1, y2) ~ x2, negbinomial, ndata, trace = TRUE)
fit2 <- rrvglm(y1 ~ x2, negbinomial(zero = NULL), data = ndata)
coef(fit1, matrix = TRUE)
vcov(fit1)
vcov(fit2)

## End(Not run)
```

---

venice *Venice Maximum Sea Levels Data*

Description

Some sea levels data sets recorded at Venice, Italy.

Usage

```
data(venice)
data(venice90)
```

Format

`venice` is a data frame with 51 observations on the following 11 variables. It concerns the maximum heights of sea levels between 1931 and 1981.

year a numeric vector.

r1,r2,r3,r4,r5,r6,r7,r8,r9,r10 numeric vectors; `r1` is the highest recorded value, `r2` is the second highest recorded value, etc.

`venice90` is a data frame with 455 observations on the following 7 variables.

year, month, day, hour numeric vectors; actual time of the recording.

sealevel numeric; sea level.

ohour numeric; number of hours since the midnight of 31 Dec 1939 and 1 Jan 1940.

Year numeric vector; approximate year as a real number. The formula is `start.year + ohour / (365.26 * 24)` where `start.year` is 1940. One can treat `Year` as continuous whereas `year` can be treated as both continuous and discrete.

Details

Sea levels are in cm. For venice90, the value 0 corresponds to a fixed reference point (e.g., the mean sea level in 1897 at an old palace of Venice). Clearly since the relative (perceived) mean sea level has been increasing in trend over time (more than an overall 0.4 m increase by 2010), therefore the value 0 is (now) a very low and unusual measurement.

For venice, in 1935 only the top six values were recorded.

For venice90, this is a subset of a data set provided by Paolo Pirazzoli consisting of hourly sea levels from 1940 to 2009. Values greater than 90 cm were extracted, and then declustered (each cluster provides no more than one value, and each value is at least 24 hours apart). Thus the values are more likely to be independent. Of the original (2009-1940+1)*365.26*24 values about 7 percent of these comprise venice90.

Yet to do: check for consistency between the data sets. Some external data sets elsewhere have some extremes recorded at times not exactly on the hour.

Source

Pirazzoli, P. (1982) Maree estreme a Venezia (periodo 1872–1981). *Acqua Aria*, **10**, 1023–1039.

Thanks to Paolo Pirazzoli and Alberto Tomasin for the venice90 data.

References

Smith, R. L. (1986). Extreme value theory based on the *r* largest annual events. *Journal of Hydrology*, **86**, 27–43.

Battistin, D. and Canestrelli, P. (2006). *La serie storica delle maree a Venezia, 1872–2004* (in Italian), Comune di Venezia. Istituzione Centro Previsione e Segnalazioni Maree.

See Also

guplot, gev, gpd.

Examples

```
## Not run:
matplot(venice[["year"]], venice[, -1], xlab = "Year",
        ylab = "Sea level (cm)", type = "l")

ymat <- as.matrix(venice[, paste("r", 1:10, sep = "")])
fit1 <- vgam(ymat ~ s(year, df = 3), gumbel(R = 365, mpv = TRUE),
             venice, trace = TRUE, na.action = na.pass)
head(fitted(fit1))

par(mfrow = c(2, 1), xpd = TRUE)
plot(fit1, se = TRUE, lcol = "blue", llwd = 2, slty = "dashed")

par(mfrow = c(1,1), bty = "l", xpd = TRUE, las = 1)
qtplot(fit1, mpv = TRUE, lcol = c(1, 2, 5), tcol = c(1, 2, 5),
       llwd = 2, pcol = "blue", tadj = 0.1)

plot(sealevel ~ Year, data = venice90, type = "h", col = "blue")
```

```
summary(venice90)
dim(venice90)
round(100 * nrow(venice90)/((2009-1940+1)*365.26*24), dig = 3)

## End(Not run)
```

vgam *Fitting Vector Generalized Additive Models*

Description

Fit a vector generalized additive model (VGAM). Both 1st-generation VGAMs (based on backfitting) and 2nd-generation VGAMs (based on P-splines, with automatic smoothing parameter selection) are implemented. This is a large class of models that includes generalized additive models (GAMs) and vector generalized linear models (VGLMs) as special cases.

Usage

```
vgam(formula,
     family = stop("argument 'family' needs to be assigned"),
     data = list(), weights = NULL, subset = NULL,
     na.action = na.fail, etastart = NULL, mustart = NULL,
     coefstart = NULL, control = vgam.control(...),
     offset = NULL, method = "vgam.fit", model = FALSE,
     x.arg = TRUE, y.arg = TRUE, contrasts = NULL,
     constraints = NULL, extra = list(), form2 = NULL,
     qr.arg = FALSE, smart = TRUE, ...)
```

Arguments

`formula` a symbolic description of the model to be fit. The RHS of the formula is applied to each linear/additive predictor, and should include at least one `sm.os` term or `sm.ps` term or `s` term. Mixing both together is not allowed. Different variables in each linear/additive predictor can be chosen by specifying constraint matrices.

`family` Same as for `vglm`.

`data` an optional data frame containing the variables in the model. By default the variables are taken from `environment(formula)`, typically the environment from which `vgam` is called.

`weights, subset, na.action`
Same as for `vglm`. Note that `subset` may be unreliable and to get around this problem it is best to use `subset` to create a new smaller data frame and feed in the smaller data frame. See below for an example. This is a bug that needs fixing.

`etastart, mustart, coefstart`
Same as for `vglm`.

`control` a list of parameters for controlling the fitting process. See `vgam.control` for details.

`method` the method to be used in fitting the model. The default (and presently only) method `vgam.fit` uses iteratively reweighted least squares (IRLS).

`constraints, model, offset`
Same as for `vglm`.

`x.arg, y.arg` logical values indicating whether the model matrix and response vector/matrix used in the fitting process should be assigned in the `x` and `y` slots. Note the model matrix is the LM model matrix; to get the VGAM model matrix type `model.matrix(vgamfit)` where `vgamfit` is a `vgam` object.

`contrasts, extra, form2, qr.arg, smart`
Same as for `vglm`.

`...` further arguments passed into `vgam.control`.

Details

A vector generalized additive model (VGAM) is loosely defined as a statistical model that is a function of M additive predictors. The central formula is given by

$$\eta_j = \sum_{k=1}^{p} f_{(j)k}(x_k)$$

where x_k is the kth explanatory variable (almost always $x_1 = 1$ for the intercept term), and $f_{(j)k}$ are smooth functions of x_k that are estimated by smoothers. The first term in the summation is just the intercept. Currently two types of smoothers are implemented: `s` represents the older and more traditional one, called a *vector (cubic smoothing spline) smoother* and is based on Yee and Wild (1996); it is more similar to the R package **gam**. The newer one is represented by `sm.os` and `sm.ps`, and these are based on O-splines and P-splines—they allow automatic smoothing parameter selection; it is more similar to the R package **mgcv**.

In the above, $j = 1, \ldots, M$ where M is finite. If all the functions are constrained to be linear then the resulting model is a vector generalized linear model (VGLM). VGLMs are best fitted with `vglm`.

Vector (cubic smoothing spline) smoothers are represented by `s()` (see `s`). Local regression via `lo()` is *not* supported. The results of `vgam` will differ from the `gam()` (in the **gam**) because `vgam()` uses a different knot selection algorithm. In general, fewer knots are chosen because the computation becomes expensive when the number of additive predictors M is large.

Second-generation VGAMs are based on the O-splines and P-splines. The latter is due to Eilers and Marx (1996). Backfitting is not required, and estimation is performed using IRLS. The function `sm.os` represents a *smart* implementation of O-splines. The function `sm.ps` represents a *smart* implementation of P-splines. Written G2-VGAMs or P-VGAMs, this methodology should not be used unless the sample size is reasonably large. Usually an UBRE predictive criterion is optimized (at each IRLS iteration) because the scale parameter for VGAMs is usually assumed to be known. This search for optimal smoothing parameters does not always converge, and neither is it totally reliable. G2-VGAMs implicitly set `criterion = "coefficients"` so that convergence occurs when the change in the regression coefficients between 2 IRLS iterations is sufficiently small. Otherwise the search for the optimal smoothing parameters might cause the log-likelihood to decrease between 2 IRLS iterations. Currently *outer iteration* is implemented, by default, rather than *performance iteration* because the latter is more easy to converge to a local solution; see Wood (2004) for details. One can use *performance iteration* by setting `Maxit.outer = 1` in `vgam.control`.

The underlying algorithm of VGAMs is IRLS. First-generation VGAMs (called G1-VGAMs) are estimated by modified vector backfitting using vector splines. O-splines are used as the basis functions for the vector (smoothing) splines, which are a lower dimensional version of natural B-splines. The function `vgam.fit()` actually does the work. The smoothing code is based on F. O'Sullivan's BART code.

A closely related methodology based on VGAMs called *constrained additive ordination* (CAO) first forms a linear combination of the explanatory variables (called *latent variables*) and then fits a GAM to these. This is implemented in the function `cao` for a very limited choice of family functions.

Value

For G1-VGAMs and G2-VGAMs, an object of class `"vgam"` or `"pvgam"` respectively (see `vgam-class` and `pvgam-class` for further information).

WARNING

For G1-VGAMs, currently `vgam` can only handle constraint matrices `cmat`, say, such that `crossprod(cmat)` is diagonal. It can be detected by `is.buggy`. VGAMs with constraint matrices that have non-orthogonal columns should be fitted with `sm.os` or `sm.ps` terms instead of `s`.

See warnings in `vglm.control`.

Note

This function can fit a wide variety of statistical models. Some of these are harder to fit than others because of inherent numerical difficulties associated with some of them. Successful model fitting benefits from cumulative experience. Varying the values of arguments in the **VGAM** family function itself is a good first step if difficulties arise, especially if initial values can be inputted. A second, more general step, is to vary the values of arguments in `vgam.control`. A third step is to make use of arguments such as `etastart`, `coefstart` and `mustart`.

Some **VGAM** family functions end in `"ff"` to avoid interference with other functions, e.g., `binomialff`, `poissonff`. This is because **VGAM** family functions are incompatible with `glm` (and also `gam()` in **gam** and `gam` in **mgcv**).

The smart prediction (`smartpred`) library is packed with the **VGAM** library.

The theory behind the scaling parameter is currently being made more rigorous, but it it should give the same value as the scale parameter for GLMs.

Author(s)

Thomas W. Yee

References

Wood, S. N. (2004). Stable and efficient multiple smoothing parameter estimation for generalized additive models. *J. Amer. Statist. Assoc.*, **99**(467): 673–686.

Yee, T. W. and Wild, C. J. (1996). Vector generalized additive models. *Journal of the Royal Statistical Society, Series B, Methodological*, **58**, 481–493.

Yee, T. W. (2008). The VGAM Package. *R News*, **8**, 28–39.

Yee, T. W. (2015). Vector Generalized Linear and Additive Models: With an Implementation in R. New York, USA: *Springer*.

Yee, T. W. (2016). Comments on "Smoothing parameter and model selection for general smooth models" by Wood, S. N. and Pya, N. and Safken, N., *J. Amer. Statist. Assoc.*, **110**(516).

See Also

is.buggy, vgam.control, vgam-class, vglmff-class, plotvgam, summaryvgam, summarypvgam, sm.os, sm.ps, s, magic, vglm, vsmooth.spline, cao.

Examples

```
# Nonparametric proportional odds model
pneumo <- transform(pneumo, let = log(exposure.time))
vgam(cbind(normal, mild, severe) ~ s(let),
     cumulative(parallel = TRUE), data = pneumo, trace = TRUE)

# Nonparametric logistic regression
hfit <- vgam(agaaus ~ s(altitude, df = 2), binomialff, hunua)
## Not run:  plot(hfit, se = TRUE)
phfit <- predict(hfit, type = "terms", raw = TRUE, se = TRUE)
names(phfit)
head(phfit$fitted)
head(phfit$se.fit)
phfit$df
phfit$sigma

# Fit two species simultaneously
hfit2 <- vgam(cbind(agaaus, kniexc) ~ s(altitude, df = c(2, 3)),
              binomialff(multiple.responses = TRUE), data = hunua)
coef(hfit2, matrix = TRUE)  # Not really interpretable
## Not run:
plot(hfit2, se = TRUE, overlay = TRUE, lcol = 3:4, scol = 3:4)
ooo <- with(hunua, order(altitude))
with(hunua, matplot(altitude[ooo], fitted(hfit2)[ooo,],
      ylim = c(0, 0.8), las = 1,type = "l", lwd = 2,
     xlab = "Altitude (m)", ylab = "Probability of presence",
     main = "Two plant species' response curves"))
with(hunua, rug(altitude))

## End(Not run)

# The 'subset' argument does not work here. Use subset() instead.
set.seed(1)
zdata <- data.frame(x2 = runif(nn <- 500))
zdata <- transform(zdata, y = rbinom(nn, 1, 0.5))
zdata <- transform(zdata, subS = runif(nn) < 0.7)
sub.zdata <- subset(zdata, subS)  # Use this instead
if (FALSE)
  fit4a <- vgam(cbind(y, y) ~ s(x2, df = 2),
                binomialff(multiple.responses = TRUE),
```

```
                 data = zdata, subset = subS)  # This fails!!!
fit4b <- vgam(cbind(y, y) ~ s(x2, df = 2),
              binomialff(multiple.responses = TRUE),
              data = sub.zdata)  # This succeeds!!!
fit4c <- vgam(cbind(y, y) ~ sm.os(x2),
              binomialff(multiple.responses = TRUE),
              data = sub.zdata)  # This succeeds!!!
## Not run: par(mfrow = c(2, 2))
plot(fit4b, se = TRUE, shade = TRUE, shcol = "pink")
plot(fit4c, se = TRUE, shade = TRUE, shcol = "pink")

## End(Not run)
```

`vgam-class` *Class "vgam"*

Description

Vector generalized additive models.

Objects from the Class

Objects can be created by calls of the form `vgam(...)`.

Slots

`nl.chisq`**:** Object of class `"numeric"`. Nonlinear chi-squared values.

`nl.df`**:** Object of class `"numeric"`. Nonlinear chi-squared degrees of freedom values.

`spar`**:** Object of class `"numeric"` containing the (scaled) smoothing parameters.

`s.xargument`**:** Object of class `"character"` holding the variable name of any `s()` terms.

`var`**:** Object of class `"matrix"` holding approximate pointwise standard error information.

`Bspline`**:** Object of class `"list"` holding the scaled (internal and boundary) knots, and the fitted B-spline coefficients. These are used for prediction.

`extra`**:** Object of class `"list"`; the `extra` argument on entry to `vglm`. This contains any extra information that might be needed by the family function.

`family`**:** Object of class `"vglmff"`. The family function.

`iter`**:** Object of class `"numeric"`. The number of IRLS iterations used.

`predictors`**:** Object of class `"matrix"` with M columns which holds the M linear predictors.

`assign`**:** Object of class `"list"`, from class `"vlm"`. This named list gives information matching the columns and the (LM) model matrix terms.

`call`**:** Object of class `"call"`, from class `"vlm"`. The matched call.

`coefficients`**:** Object of class `"numeric"`, from class `"vlm"`. A named vector of coefficients.

`constraints`**:** Object of class `"list"`, from class `"vlm"`. A named list of constraint matrices used in the fitting.

`contrasts`: Object of class `"list"`, from class `"vlm"`. The contrasts used (if any).

`control`: Object of class `"list"`, from class `"vlm"`. A list of parameters for controlling the fitting process. See `vglm.control` for details.

`criterion`: Object of class `"list"`, from class `"vlm"`. List of convergence criterion evaluated at the final IRLS iteration.

`df.residual`: Object of class `"numeric"`, from class `"vlm"`. The residual degrees of freedom.

`df.total`: Object of class `"numeric"`, from class `"vlm"`. The total degrees of freedom.

`dispersion`: Object of class `"numeric"`, from class `"vlm"`. The scaling parameter.

`effects`: Object of class `"numeric"`, from class `"vlm"`. The effects.

`fitted.values`: Object of class `"matrix"`, from class `"vlm"`. The fitted values. This is usually the mean but may be quantiles, or the location parameter, e.g., in the Cauchy model.

`misc`: Object of class `"list"`, from class `"vlm"`. A named list to hold miscellaneous parameters.

`model`: Object of class `"data.frame"`, from class `"vlm"`. The model frame.

`na.action`: Object of class `"list"`, from class `"vlm"`. A list holding information about missing values.

`offset`: Object of class `"matrix"`, from class `"vlm"`. If non-zero, a M-column matrix of offsets.

`post`: Object of class `"list"`, from class `"vlm"` where post-analysis results may be put.

`preplot`: Object of class `"list"`, from class `"vlm"` used by `plotvgam`; the plotting parameters may be put here.

`prior.weights`: Object of class `"matrix"`, from class `"vlm"` holding the initially supplied weights.

`qr`: Object of class `"list"`, from class `"vlm"`. QR decomposition at the final iteration.

`R`: Object of class `"matrix"`, from class `"vlm"`. The **R** matrix in the QR decomposition used in the fitting.

`rank`: Object of class `"integer"`, from class `"vlm"`. Numerical rank of the fitted model.

`residuals`: Object of class `"matrix"`, from class `"vlm"`. The *working* residuals at the final IRLS iteration.

`ResSS`: Object of class `"numeric"`, from class `"vlm"`. Residual sum of squares at the final IRLS iteration with the adjusted dependent vectors and weight matrices.

`smart.prediction`: Object of class `"list"`, from class `"vlm"`. A list of data-dependent parameters (if any) that are used by smart prediction.

`terms`: Object of class `"list"`, from class `"vlm"`. The `terms` object used.

`weights`: Object of class `"matrix"`, from class `"vlm"`. The weight matrices at the final IRLS iteration. This is in matrix-band form.

`x`: Object of class `"matrix"`, from class `"vlm"`. The model matrix (LM, not VGLM).

`xlevels`: Object of class `"list"`, from class `"vlm"`. The levels of the factors, if any, used in fitting.

`y`: Object of class `"matrix"`, from class `"vlm"`. The response, in matrix form.

`Xm2`: Object of class `"matrix"`, from class `"vlm"`. See `vglm-class`).

`Ym2`: Object of class `"matrix"`, from class `"vlm"`. See `vglm-class`).

`callXm2`: Object of class `"call"`, from class `"vlm"`. The matched call for argument `form2`.

Extends

Class "vglm", directly. Class "vlm", by class "vglm".

Methods

cdf signature(object = "vglm"): cumulative distribution function. Useful for quantile regression and extreme value data models.

deplot signature(object = "vglm"): density plot. Useful for quantile regression models.

deviance signature(object = "vglm"): deviance of the model (where applicable).

plot signature(x = "vglm"): diagnostic plots.

predict signature(object = "vglm"): extract the additive predictors or predict the additive predictors at a new data frame.

print signature(x = "vglm"): short summary of the object.

qtplot signature(object = "vglm"): quantile plot (only applicable to some models).

resid signature(object = "vglm"): residuals. There are various types of these.

residuals signature(object = "vglm"): residuals. Shorthand for resid.

rlplot signature(object = "vglm"): return level plot. Useful for extreme value data models.

summary signature(object = "vglm"): a more detailed summary of the object.

Note

VGAMs have all the slots that vglm objects have (vglm-class), plus the first few slots described in the section above.

Author(s)

Thomas W. Yee

References

Yee, T. W. and Wild, C. J. (1996). Vector generalized additive models. *Journal of the Royal Statistical Society, Series B, Methodological*, **58**, 481–493.

See Also

vgam.control, vglm, s, vglm-class, vglmff-class.

Examples

```
# Fit a nonparametric proportional odds model
pneumo <- transform(pneumo, let = log(exposure.time))
vgam(cbind(normal, mild, severe) ~ s(let),
     cumulative(parallel = TRUE), data = pneumo)
```

`vgam.control` *Control Function for vgam()*

Description

Algorithmic constants and parameters for running `vgam` are set using this function.

Usage

```
vgam.control(all.knots = FALSE, bf.epsilon = 1e-07, bf.maxit = 30,
             checkwz=TRUE, Check.rank = TRUE, Check.cm.rank = TRUE,
             criterion = names(.min.criterion.VGAM),
             epsilon = 1e-07, maxit = 30, Maxit.outer = 10,
             noWarning = FALSE,
             na.action = na.fail,
             nk = NULL, save.weights = FALSE, se.fit = TRUE,
             trace = FALSE, wzepsilon = .Machine$double.eps^0.75,
             xij = NULL, gamma.arg = 1, ...)
```

Arguments

`all.knots` logical indicating if all distinct points of the smoothing variables are to be used as knots. By default, `all.knots=TRUE` for $n \leq 40$, and for $n > 40$, the number of knots is approximately $40 + (n - 40)^{0.25}$. This increases very slowly with n so that the number of knots is approximately between 50 and 60 for large n.

`bf.epsilon` tolerance used by the modified vector backfitting algorithm for testing convergence. Must be a positive number.

`bf.maxit` maximum number of iterations allowed in the modified vector backfitting algorithm. Must be a positive integer.

`checkwz` logical indicating whether the diagonal elements of the working weight matrices should be checked whether they are sufficiently positive, i.e., greater than `wzepsilon`. If not, any values less than `wzepsilon` are replaced with this value.

`Check.rank, Check.cm.rank` See `vglm.control`.

`criterion` character variable describing what criterion is to be used to test for convergence. The possibilities are listed in `.min.criterion.VGAM`, but most family functions only implement a few of these.

`epsilon` positive convergence tolerance epsilon. Roughly speaking, the Newton-Raphson/Fisher-scoring/local-scoring iterations are assumed to have converged when two successive `criterion` values are within `epsilon` of each other.

`maxit` maximum number of Newton-Raphson/Fisher-scoring/local-scoring iterations allowed.

`Maxit.outer` maximum number of outer iterations allowed when there are `sm.os` or `sm.ps` terms. See `vgam` for a little information about the default *outer iteration*. Note that one can use *performance iteration* by setting `Maxit.outer = 1`; then the smoothing parameters will be automatically chosen at each IRLS iteration (some specific programming allows this).

Note that `gam` uses outer iteration by default. However, `magic` is only invoked for the Gaussian family, so the results of `gam` may differ substantially from `sm.os` and `sm.ps` in general.

`na.action` how to handle missing values. Unlike the SPLUS `gam` function, `vgam` cannot handle `NA`s when smoothing.

`nk` vector of length d containing positive integers. where d be the number of `s` terms in the formula. Recycling is used if necessary. The ith value is the number of B-spline coefficients to be estimated for each component function of the ith `s()` term. `nk` differs from the number of knots by some constant. If specified, `nk` overrides the automatic knot selection procedure.

`save.weights` logical indicating whether the `weights` slot of a `"vglm"` object will be saved on the object. If not, it will be reconstructed when needed, e.g., `summary`.

`se.fit` logical indicating whether approximate pointwise standard errors are to be saved on the object. If `TRUE`, then these can be plotted with `plot(..., se = TRUE)`.

`trace` logical indicating if output should be produced for each iteration.

`wzepsilon` Small positive number used to test whether the diagonals of the working weight matrices are sufficiently positive.

`noWarning` Same as `vglm.control`.

`xij` Same as `vglm.control`.

`gamma.arg` Numeric; same as `gamma` in `magic`. Inflation factor for optimizing the UBRE/GCV criterion. If given, a suggested value is 1.4 to help avoid overfitting, based on the work of Gu and co-workers (values between 1.2 and 1.4 appeared reasonable, based on simulations). A warning may be given if the value is deemed out-of-range.

`...` other parameters that may be picked up from control functions that are specific to the **VGAM** family function.

Details

Most of the control parameters are used within `vgam.fit` and you will have to look at that to understand the full details. Many of the control parameters are used in a similar manner by `vglm.fit` (`vglm`) because the algorithm (IRLS) is very similar.

Setting `save.weights=FALSE` is useful for some models because the `weights` slot of the object is often the largest and so less memory is used to store the object. However, for some **VGAM** family function, it is necessary to set `save.weights=TRUE` because the `weights` slot cannot be reconstructed later.

Value

A list with components matching the input names. A little error checking is done, but not much. The list is assigned to the `control` slot of `vgam` objects.

Warning

See vglm.control.

Note

vgam does not implement half-stepsizing, therefore parametric models should be fitted with vglm. Also, vgam is slower than vglm too.

Author(s)

Thomas W. Yee

References

Yee, T. W. and Wild, C. J. (1996). Vector generalized additive models. *Journal of the Royal Statistical Society, Series B, Methodological*, **58**, 481–493.

See Also

vgam, vglm.control, vsmooth.spline, vglm.

Examples

```
pneumo <- transform(pneumo, let = log(exposure.time))
vgam(cbind(normal, mild, severe) ~ s(let, df = 2), multinomial,
     data = pneumo, trace = TRUE, eps = 1e-4, maxit = 10)
```

---

vglm *Fitting Vector Generalized Linear Models*

Description

vglm fits vector generalized linear models (VGLMs). This very large class of models includes generalized linear models (GLMs) as a special case.

Usage

```
vglm(formula,
     family = stop("argument 'family' needs to be assigned"),
     data = list(), weights = NULL, subset = NULL,
     na.action = na.fail, etastart = NULL, mustart = NULL,
     coefstart = NULL, control = vglm.control(...), offset = NULL,
     method = "vglm.fit", model = FALSE, x.arg = TRUE, y.arg = TRUE,
     contrasts = NULL, constraints = NULL, extra = list(),
     form2 = NULL, qr.arg = TRUE, smart = TRUE, ...)
```

Arguments

formula
: a symbolic description of the model to be fit. The RHS of the formula is applied to each linear predictor. The effect of different variables in each linear predictor can be controlled by specifying constraint matrices—see `constraints` below.

family
: a function of class "vglmff" (see `vglmff-class`) describing what statistical model is to be fitted. This is called a "**VGAM** family function". See `CommonVGAMffArguments` for general information about many types of arguments found in this type of function. The argument name "family" is used loosely and for the ease of existing `glm` users; there is no concept of a formal "error distribution" for VGLMs. Possibly the argument name should be better "model" but unfortunately that name has already been taken.

data
: an optional data frame containing the variables in the model. By default the variables are taken from `environment(formula)`, typically the environment from which `vglm` is called.

weights
: an optional vector or matrix of (prior fixed and known) weights to be used in the fitting process. If the **VGAM** family function handles multiple responses ($Q > 1$ of them, say) then `weights` can be a matrix with Q columns. Each column matches the respective response. If it is a vector (the usually case) then it is recycled into a matrix with Q columns. The values of `weights` must be positive; try setting a very small value such as `1.0e-8` to effectively delete an observation.

  Currently the `weights` argument supports sampling weights from complex sampling designs via **svyVGAM**. Some details can be found at https://CRAN.R-project.org/package=svyVGAM.

subset
: an optional logical vector specifying a subset of observations to be used in the fitting process.

na.action
: a function which indicates what should happen when the data contain NAs. The default is set by the `na.action` setting of `options`, and is `na.fail` if that is unset. The "factory-fresh" default is `na.omit`.

etastart
: optional starting values for the linear predictors. It is a M-column matrix with the same number of rows as the response. If $M = 1$ then it may be a vector. Note that `etastart` and the output of `predict(fit)` should be comparable. Here, `fit` is the fitted object. Almost all **VGAM** family functions are self-starting.

mustart
: optional starting values for the fitted values. It can be a vector or a matrix; if a matrix, then it has the same number of rows as the response. Usually `mustart` and the output of `fitted(fit)` should be comparable. Most family functions do not make use of this argument because it is not possible to compute all M columns of `eta` from `mu`.

coefstart
: optional starting values for the coefficient vector. The length and order must match that of `coef(fit)`.

control
: a list of parameters for controlling the fitting process. See `vglm.control` for details.

offset
: a vector or M-column matrix of offset values. These are *a priori* known and are added to the linear/additive predictors during fitting.

method — the method to be used in fitting the model. The default (and presently only) method `vglm.fit()` uses iteratively reweighted least squares (IRLS).

model — a logical value indicating whether the *model frame* should be assigned in the `model` slot.

x.arg, y.arg — logical values indicating whether the LM matrix and response vector/matrix used in the fitting process should be assigned in the `x` and `y` slots. Note that the model matrix is the LM matrix; to get the VGLM matrix type `model.matrix(vglmfit)` where `vglmfit` is a `vglm` object.

contrasts — an optional list. See the `contrasts.arg` of `model.matrix.default`.

constraints — an optional `list` of constraint matrices. The components of the list must be named (labelled) with the term it corresponds to (and it must match in character format *exactly*—see below). There are two types of input: `"lm"`-type and `"vlm"`-type. The former is a subset of the latter. The former has a matrix for each term of the LM matrix. The latter has a matrix for each column of the big VLM matrix. After fitting, the `constraints` extractor function may be applied; it returns the `"vlm"`-type list of constraint matrices by default. If `"lm"`-type are returned by `constraints` then these can be fed into this argument and it should give the same model as before.

If the `constraints` argument is used then the family function's `zero` argument (if it exists) needs to be set to `NULL`. This avoids what could be a probable contradiction. Sometimes setting other arguments related to constraint matrices to `FALSE` is also a good idea, e.g., `parallel = FALSE`, `exchangeable = FALSE`.

Properties: each constraint matrix must have M rows, and be of full-column rank. By default, constraint matrices are the M by M identity matrix unless arguments in the family function itself override these values, e.g., `parallel` (see `CommonVGAMffArguments`). If `constraints` is used then it must contain *all* the terms; an incomplete list is not accepted.

As mentioned above, the labelling of each constraint matrix must match exactly, e.g., `list("s(x2,df=3)"=diag(2))` will fail as `as.character(~ s(x2,df=3))` produces white spaces: `"s(x2, df = 3)"`. Thus `list("s(x2, df = 3)" = diag(2))` is needed. See Example 6 below. More details are given in Yee (2015; Section 3.3.1.3) which is on p.101. Note that the label for the intercept is `"(Intercept)"`.

extra — an optional list with any extra information that might be needed by the **VGAM** family function.

form2 — the second (optional) formula. If argument `xij` is used (see `vglm.control`) then `form2` needs to have *all* terms in the model. Also, some **VGAM** family functions such as `micmen` use this argument to input the regressor variable. If given, the slots `@Xm2` and `@Ym2` may be assigned. Note that smart prediction applies to terms in `form2` too.

qr.arg — logical value indicating whether the slot `qr`, which returns the QR decomposition of the VLM model matrix, is returned on the object.

smart — logical value indicating whether smart prediction (`smartpred`) will be used.

... — further arguments passed into `vglm.control`.

Details

A vector generalized linear model (VGLM) is loosely defined as a statistical model that is a function of M linear predictors and can be estimated by Fisher scoring. The central formula is given by

$$\eta_j = \beta_j^T x$$

where x is a vector of explanatory variables (sometimes just a 1 for an intercept), and β_j is a vector of regression coefficients to be estimated. Here, $j = 1, \ldots, M$, where M is finite. Then one can write $\eta = (\eta_1, \ldots, \eta_M)^T$ as a vector of linear predictors.

Most users will find `vglm` similar in flavour to `glm`. The function `vglm.fit` actually does the work.

Value

An object of class `"vglm"`, which has the following slots. Some of these may not be assigned to save space, and will be recreated if necessary later.

`extra`	the list `extra` at the end of fitting.
`family`	the family function (of class `"vglmff"`).
`iter`	the number of IRLS iterations used.
`predictors`	a M-column matrix of linear predictors.
`assign`	a named list which matches the columns and the (LM) model matrix terms.
`call`	the matched call.
`coefficients`	a named vector of coefficients.
`constraints`	a named list of constraint matrices used in the fitting.
`contrasts`	the contrasts used (if any).
`control`	list of control parameter used in the fitting.
`criterion`	list of convergence criterion evaluated at the final IRLS iteration.
`df.residual`	the residual degrees of freedom.
`df.total`	the total degrees of freedom.
`dispersion`	the scaling parameter.
`effects`	the effects.
`fitted.values`	the fitted values, as a matrix. This is often the mean but may be quantiles, or the location parameter, e.g., in the Cauchy model.
`misc`	a list to hold miscellaneous parameters.
`model`	the model frame.
`na.action`	a list holding information about missing values.
`offset`	if non-zero, a M-column matrix of offsets.
`post`	a list where post-analysis results may be put.
`preplot`	used by `plotvgam`, the plotting parameters may be put here.
`prior.weights`	initially supplied weights (the `weights` argument). Also see `weightsvglm`.
`qr`	the QR decomposition used in the fitting.

`R`	the **R** matrix in the QR decomposition used in the fitting.
`rank`	numerical rank of the fitted model.
`residuals`	the *working* residuals at the final IRLS iteration.
`ResSS`	residual sum of squares at the final IRLS iteration with the adjusted dependent vectors and weight matrices.
`smart.prediction`	a list of data-dependent parameters (if any) that are used by smart prediction.
`terms`	the `terms` object used.
`weights`	the working weight matrices at the final IRLS iteration. This is in matrix-band form.
`x`	the model matrix (linear model LM, not VGLM).
`xlevels`	the levels of the factors, if any, used in fitting.
`y`	the response, in matrix form.

This slot information is repeated at `vglm-class`.

WARNING

See warnings in `vglm.control`. Also, see warnings under `weights` above regarding sampling weights from complex sampling designs.

Note

This function can fit a wide variety of statistical models. Some of these are harder to fit than others because of inherent numerical difficulties associated with some of them. Successful model fitting benefits from cumulative experience. Varying the values of arguments in the **VGAM** family function itself is a good first step if difficulties arise, especially if initial values can be inputted. A second, more general step, is to vary the values of arguments in `vglm.control`. A third step is to make use of arguments such as `etastart`, `coefstart` and `mustart`.

Some **VGAM** family functions end in `"ff"` to avoid interference with other functions, e.g., `binomialff`, `poissonff`. This is because **VGAM** family functions are incompatible with `glm` (and also `gam()` in **gam** and `gam` in the **mgcv** library).

The smart prediction (`smartpred`) library is incorporated within the **VGAM** library.

The theory behind the scaling parameter is currently being made more rigorous, but it it should give the same value as the scale parameter for GLMs.

In Example 5 below, the `xij` argument to illustrate covariates that are specific to a linear predictor. Here, `lop`/`rop` are the ocular pressures of the left/right eye (artificial data). Variables `leye` and `reye` might be the presence/absence of a particular disease on the LHS/RHS eye respectively. See `vglm.control` and `fill1` for more details and examples.

Author(s)

Thomas W. Yee

References

Yee, T. W. (2015). Vector Generalized Linear and Additive Models: With an Implementation in R. New York, USA: *Springer*.

Yee, T. W. and Hastie, T. J. (2003). Reduced-rank vector generalized linear models. *Statistical Modelling*, **3**, 15–41.

Yee, T. W. and Wild, C. J. (1996). Vector generalized additive models. *Journal of the Royal Statistical Society, Series B, Methodological*, **58**, 481–493.

Yee, T. W. (2014). Reduced-rank vector generalized linear models with two linear predictors. *Computational Statistics and Data Analysis*, **71**, 889–902.

Yee, T. W. (2008). The VGAM Package. *R News*, **8**, 28–39.

See Also

vglm.control, vglm-class, vglmff-class, smartpred, vglm.fit, fill1, rrvglm, vgam. Methods functions include add1.vglm, anova.vglm, AICvlm, coefvlm, confintvglm, constraints.vlm, drop1.vglm, fittedvlm, hatvaluesvlm, hdeff.vglm, Influence.vglm, linkfunvlm, lrt.stat.vlm, score.stat.vlm, wald.stat.vlm, nobs.vlm, npred.vlm, plotvglm, predictvglm, residualsvglm, step4vglm, summaryvglm, lrtest_vglm, update, TypicalVGAMfamilyFunction, etc.

Examples

```
# Example 1. See help(glm)
(d.AD <- data.frame(treatment = gl(3, 3),
                    outcome = gl(3, 1, 9),
                    counts = c(18,17,15,20,10,20,25,13,12)))
vglm.D93 <- vglm(counts ~ outcome + treatment, poissonff,
                 data = d.AD, trace = TRUE)
summary(vglm.D93)


# Example 2. Multinomial logit model
pneumo <- transform(pneumo, let = log(exposure.time))
vglm(cbind(normal, mild, severe) ~ let, multinomial, pneumo)


# Example 3. Proportional odds model
fit3 <- vglm(cbind(normal, mild, severe) ~ let, propodds, pneumo)
coef(fit3, matrix = TRUE)
constraints(fit3)
model.matrix(fit3, type = "lm")  # LM model matrix
model.matrix(fit3)               # Larger VGLM (or VLM) matrix


# Example 4. Bivariate logistic model
fit4 <- vglm(cbind(nBnW, nBW, BnW, BW) ~ age, binom2.or, coalminers)
coef(fit4, matrix = TRUE)
depvar(fit4)  # Response are proportions
weights(fit4, type = "prior")
```

```
# Example 5. The use of the xij argument (simple case).
# The constraint matrix for 'op' has one column.
nn <- 1000
eyesdat <- round(data.frame(lop = runif(nn),
                            rop = runif(nn),
                             op = runif(nn)), digits = 2)
eyesdat <- transform(eyesdat, eta1 = -1 + 2 * lop,
                              eta2 = -1 + 2 * lop)
eyesdat <- transform(eyesdat,
           leye = rbinom(nn, 1, prob = logitlink(eta1, inv = TRUE)),
           reye = rbinom(nn, 1, prob = logitlink(eta2, inv = TRUE)))
head(eyesdat)
fit5 <- vglm(cbind(leye, reye) ~ op,
             binom2.or(exchangeable = TRUE, zero = 3),
             data = eyesdat, trace = TRUE,
             xij = list(op ~ lop + rop + fill1(lop)),
             form2 = ~  op + lop + rop + fill1(lop))
coef(fit5)
coef(fit5, matrix = TRUE)
constraints(fit5)
fit5@control$xij
head(model.matrix(fit5))


# Example 6. The use of the 'constraints' argument.
as.character(~ bs(year,df=3))  # Get the white spaces right
clist <- list("(Intercept)"      = diag(3),
              "bs(year, df = 3)" = rbind(1, 0, 0))
fit1 <- vglm(r1 ~ bs(year,df=3), gev(zero = NULL),
             data = venice, constraints = clist, trace = TRUE)
coef(fit1, matrix = TRUE)  # Check
```

---

vglm-class *Class "vglm"*

Description

Vector generalized linear models.

Objects from the Class

Objects can be created by calls of the form `vglm(...)`.

Slots

In the following, M is the number of linear predictors.

`extra`**:** Object of class `"list"`; the `extra` argument on entry to `vglm`. This contains any extra information that might be needed by the family function.

`family`: Object of class "`vglmff`". The family function.

`iter`: Object of class "`numeric`". The number of IRLS iterations used.

`predictors`: Object of class "`matrix`" with M columns which holds the M linear predictors.

`assign`: Object of class "`list`", from class "`vlm`". This named list gives information matching the columns and the (LM) model matrix terms.

`call`: Object of class "`call`", from class "`vlm`". The matched call.

`coefficients`: Object of class "`numeric`", from class "`vlm`". A named vector of coefficients.

`constraints`: Object of class "`list`", from class "`vlm`". A named list of constraint matrices used in the fitting.

`contrasts`: Object of class "`list`", from class "`vlm`". The contrasts used (if any).

`control`: Object of class "`list`", from class "`vlm`". A list of parameters for controlling the fitting process. See `vglm.control` for details.

`criterion`: Object of class "`list`", from class "`vlm`". List of convergence criterion evaluated at the final IRLS iteration.

`df.residual`: Object of class "`numeric`", from class "`vlm`". The residual degrees of freedom.

`df.total`: Object of class "`numeric`", from class "`vlm`". The total degrees of freedom.

`dispersion`: Object of class "`numeric`", from class "`vlm`". The scaling parameter.

`effects`: Object of class "`numeric`", from class "`vlm`". The effects.

`fitted.values`: Object of class "`matrix`", from class "`vlm`". The fitted values.

`misc`: Object of class "`list`", from class "`vlm`". A named list to hold miscellaneous parameters.

`model`: Object of class "`data.frame`", from class "`vlm`". The model frame.

`na.action`: Object of class "`list`", from class "`vlm`". A list holding information about missing values.

`offset`: Object of class "`matrix`", from class "`vlm`". If non-zero, a M-column matrix of offsets.

`post`: Object of class "`list`", from class "`vlm`" where post-analysis results may be put.

`preplot`: Object of class "`list`", from class "`vlm`" used by `plotvgam`; the plotting parameters may be put here.

`prior.weights`: Object of class "`matrix`", from class "`vlm`" holding the initially supplied weights.

`qr`: Object of class "`list`", from class "`vlm`". QR decomposition at the final iteration.

`R`: Object of class "`matrix`", from class "`vlm`". The **R** matrix in the QR decomposition used in the fitting.

`rank`: Object of class "`integer`", from class "`vlm`". Numerical rank of the fitted model.

`residuals`: Object of class "`matrix`", from class "`vlm`". The *working* residuals at the final IRLS iteration.

`ResSS`: Object of class "`numeric`", from class "`vlm`". Residual sum of squares at the final IRLS iteration with the adjusted dependent vectors and weight matrices.

`smart.prediction`: Object of class "`list`", from class "`vlm`". A list of data-dependent parameters (if any) that are used by smart prediction.

`terms`: Object of class "`list`", from class "`vlm`". The `terms` object used.

`weights`: Object of class "matrix", from class "vlm". The weight matrices at the final IRLS iteration. This is in matrix-band form.

`x`: Object of class "matrix", from class "vlm". The model matrix (LM, not VGLM).

`xlevels`: Object of class "list", from class "vlm". The levels of the factors, if any, used in fitting.

`y`: Object of class "matrix", from class "vlm". The response, in matrix form.

`Xm2`: Object of class "matrix", from class "vlm". See vglm-class).

`Ym2`: Object of class "matrix", from class "vlm". See vglm-class).

`callXm2`: Object of class "call", from class "vlm". The matched call for argument `form2`.

Extends

Class "vlm", directly.

Methods

cdf `signature(object = "vglm")`: cumulative distribution function. Applicable to, e.g., quantile regression and extreme value data models.

deplot `signature(object = "vglm")`: Applicable to, e.g., quantile regression.

deviance `signature(object = "vglm")`: deviance of the model (where applicable).

plot `signature(x = "vglm")`: diagnostic plots.

predict `signature(object = "vglm")`: extract the linear predictors or predict the linear predictors at a new data frame.

print `signature(x = "vglm")`: short summary of the object.

qtplot `signature(object = "vglm")`: quantile plot (only applicable to some models).

resid `signature(object = "vglm")`: residuals. There are various types of these.

residuals `signature(object = "vglm")`: residuals. Shorthand for `resid`.

rlplot `signature(object = "vglm")`: return level plot. Useful for extreme value data models.

summary `signature(object = "vglm")`: a more detailed summary of the object.

Author(s)

Thomas W. Yee

References

Yee, T. W. and Hastie, T. J. (2003). Reduced-rank vector generalized linear models. *Statistical Modelling*, **3**, 15–41.

Yee, T. W. and Wild, C. J. (1996). Vector generalized additive models. *Journal of the Royal Statistical Society, Series B, Methodological*, **58**, 481–493.

See Also

vglm, vglmff-class, vgam-class.

Examples

```
# Multinomial logit model
pneumo <- transform(pneumo, let = log(exposure.time))
vglm(cbind(normal, mild, severe) ~ let, multinomial, data = pneumo)
```

`vglm.control` *Control Function for vglm()*

Description

Algorithmic constants and parameters for running `vglm` are set using this function.

Usage

```
vglm.control(checkwz = TRUE, Check.rank = TRUE, Check.cm.rank = TRUE,
             criterion = names(.min.criterion.VGAM),
             epsilon = 1e-07, half.stepsizing = TRUE,
             maxit = 30, noWarning = FALSE,
             stepsize = 1, save.weights = FALSE,
             trace = FALSE, wzepsilon = .Machine$double.eps^0.75,
             xij = NULL, ...)
```

Arguments

`checkwz` logical indicating whether the diagonal elements of the working weight matrices should be checked whether they are sufficiently positive, i.e., greater than `wzepsilon`. If not, any values less than `wzepsilon` are replaced with this value.

`Check.rank` logical indicating whether the rank of the VLM matrix should be checked. If this is not of full column rank then the results are not to be trusted. The default is to give an error message if the VLM matrix is not of full column rank.

`Check.cm.rank` logical indicating whether the rank of each constraint matrix should be checked. If this is not of full column rank then an error will occur. Under no circumstances should any constraint matrix have a rank less than the number of columns.

`criterion` character variable describing what criterion is to be used to test for convergence. The possibilities are listed in `.min.criterion.VGAM`, but most family functions only implement a few of these.

`epsilon` positive convergence tolerance epsilon. Roughly speaking, the Newton-Raphson/Fisher-scoring iterations are assumed to have converged when two successive `criterion` values are within `epsilon` of each other.

`half.stepsizing`
logical indicating if half-stepsizing is allowed. For example, in maximizing a log-likelihood, if the next iteration has a log-likelihood that is less than the current value of the log-likelihood, then a half step will be taken. If the log-likelihood is still less than at the current position, a quarter-step will be taken etc. Eventually a step will be taken so that an improvement is made to the convergence criterion. `half.stepsizing` is ignored if `criterion == "coefficients"`.

`maxit` maximum number of (usually Fisher-scoring) iterations allowed. Sometimes Newton-Raphson is used.

`noWarning` logical indicating whether to suppress a warning if convergence is not obtained within `maxit` iterations. This is ignored if `maxit = 1` is set.

`stepsize` usual step size to be taken between each Newton-Raphson/Fisher-scoring iteration. It should be a value between 0 and 1, where a value of unity corresponds to an ordinary step. A value of 0.5 means half-steps are taken. Setting a value near zero will cause convergence to be generally slow but may help increase the chances of successful convergence for some family functions.

`save.weights` logical indicating whether the `weights` slot of a `"vglm"` object will be saved on the object. If not, it will be reconstructed when needed, e.g., `summary`. Some family functions have `save.weights = TRUE` and others have `save.weights = FALSE` in their control functions.

`trace` logical indicating if output should be produced for each iteration. Setting `trace = TRUE` is recommended in general because **VGAM** fits a very broad variety of models and distributions, and for some of them, convergence is intrinsically more difficult. Monitoring convergence can help check that the solution is reasonable or that a problem has occurred. It may suggest better initial values are needed, the making of invalid assumptions, or that the model is inappropriate for the data, etc.

`wzepsilon` small positive number used to test whether the diagonals of the working weight matrices are sufficiently positive.

`xij` A list of formulas. Each formula has a RHS giving M terms making up a covariate-dependent term (whose name is the response). That is, it creates a variable that takes on different values for each linear/additive predictor, e.g., the ocular pressure of each eye. The M terms must be unique; use `fill1`, `fill2`, `fill3`, etc. if necessary. Each formula should have a response which is taken as the name of that variable, and the M terms are enumerated in sequential order. Each of the M terms multiply each successive row of the constraint matrix. When `xij` is used, the use of `form2` is also required to give *every* term used by the model.

A formula or a list of formulas.

The function `Select` can be used to select variables beginning with the same character string.

`...` other parameters that may be picked up from control functions that are specific to the **VGAM** family function.

Details

Most of the control parameters are used within `vglm.fit` and you will have to look at that to understand the full details.

Setting `save.weights = FALSE` is useful for some models because the `weights` slot of the object is the largest and so less memory is used to store the object. However, for some **VGAM** family function, it is necessary to set `save.weights = TRUE` because the `weights` slot cannot be reconstructed later.

Value

A list with components matching the input names. A little error checking is done, but not much. The list is assigned to the `control` slot of `vglm` objects.

Warning

For some applications the default convergence criterion should be tightened. Setting something like `criterion = "coef", epsilon = 1e-09` is one way to achieve this, and also add `trace = TRUE` to monitor the convergence. Setting `maxit` to some higher number is usually not needed, and needing to do so suggests something is wrong, e.g., an ill-conditioned model, over-fitting or under-fitting.

Note

Reiterating from above, setting `trace = TRUE` is recommended in general.

In Example 2 below there are two covariates that have linear/additive predictor specific values. These are handled using the `xij` argument.

Author(s)

Thomas W. Yee

References

Yee, T. W. and Hastie, T. J. (2003). Reduced-rank vector generalized linear models. *Statistical Modelling*, **3**, 15–41.

See Also

`vglm`, `TypicalVGAMfamilyFunction`, `fill1`. The author's homepage has further documentation about the `xij` argument; see also `Select`.

Examples

```
# Example 1.
pneumo <- transform(pneumo, let = log(exposure.time))
vglm(cbind(normal, mild, severe) ~ let, multinomial, pneumo,
     crit = "coef", step = 0.5, trace = TRUE, epsil = 1e-8,
     maxit = 40)


# Example 2. The use of the xij argument (simple case).
ymat <- rdiric(n <- 1000, shape = rep(exp(2), len = 4))
mydat <- data.frame(x1 = runif(n), x2 = runif(n), x3 = runif(n),
                    x4 = runif(n),
                    z1 = runif(n), z2 = runif(n), z3 = runif(n),
                    z4 = runif(n))
mydat <- transform(mydat, X = x1, Z = z1)
mydat <- round(mydat, digits = 2)
fit2 <- vglm(ymat ~ X + Z,
             dirichlet(parallel = TRUE), mydat, trace = TRUE,
```

```
              xij = list(Z ~ z1 + z2 + z3 + z4,
                         X ~ x1 + x2 + x3 + x4),
              form2 = ~  Z + z1 + z2 + z3 + z4 +
                         X + x1 + x2 + x3 + x4)
head(model.matrix(fit2, type =  "lm"))  # LM model matrix
head(model.matrix(fit2, type = "vlm"))  # Big VLM model matrix
coef(fit2)
coef(fit2, matrix = TRUE)
max(abs(predict(fit2)-predict(fit2, new = mydat)))  # Predicts ok
summary(fit2)
## Not run:
# plotvgam(fit2, se = TRUE, xlab = "x1", which.term = 1)  # Bug!
# plotvgam(fit2, se = TRUE, xlab = "z1", which.term = 2)  # Bug!
plotvgam(fit2, xlab = "x1")  # Correct
plotvgam(fit2, xlab = "z1")  # Correct

## End(Not run)


# Example 3. The use of the xij argument (complex case).
set.seed(123)
coalminers <-
  transform(coalminers,
            Age = (age - 42) / 5,
            dum1 = round(runif(nrow(coalminers)), digits = 2),
            dum2 = round(runif(nrow(coalminers)), digits = 2),
            dum3 = round(runif(nrow(coalminers)), digits = 2),
            dumm = round(runif(nrow(coalminers)), digits = 2))
BS <- function(x, ..., df = 3)
  sm.bs(c(x,...), df = df)[1:length(x),,drop = FALSE]
NS <- function(x, ..., df = 3)
  sm.ns(c(x,...), df = df)[1:length(x),,drop = FALSE]

# Equivalently...
BS <- function(x, ..., df = 3)
  head(sm.bs(c(x,...), df = df), length(x), drop = FALSE)
NS <- function(x, ..., df = 3)
  head(sm.ns(c(x,...), df = df), length(x), drop = FALSE)

fit3 <- vglm(cbind(nBnW,nBW,BnW,BW) ~ Age + NS(dum1, dum2),
             fam = binom2.or(exchangeable = TRUE, zero = 3),
             xij = list(NS(dum1, dum2) ~ NS(dum1, dum2) +
                                         NS(dum2, dum1) +
                                         fill1(NS( dum1))),
             form2 = ~  NS(dum1, dum2) + NS(dum2, dum1) +
                        fill1(NS(dum1)) +
                        dum1 + dum2 + dum3 + Age + age + dumm,
             data = coalminers, trace = TRUE)
head(model.matrix(fit3, type = "lm"))   # LM model matrix
head(model.matrix(fit3, type = "vlm"))  # Big VLM model matrix
coef(fit3)
coef(fit3, matrix = TRUE)
## Not run:
```

```
plotvgam(fit3, se = TRUE, lcol = 2, scol = 4, xlab = "dum1")

## End(Not run)
```

---

vglmff-class *Class "vglmff"*

Description

Family functions for the **VGAM** package

Objects from the Class

Objects can be created by calls of the form `new("vglmff", ...)`.

Slots

In the following, M is the number of linear/additive predictors.

`start1`: Object of class `"expression"` to insert code at a special position (the very start) in `vglm.fit` or `vgam.fit`.

`blurb`: Object of class `"character"` giving a small description of the model. Important arguments such as parameter link functions can be expressed here.

`charfun`: Object of class `"function"` which returns the characteristic function or variance function (usually for some GLMs only). The former uses a dummy variable x. Both use the linear/additive predictors. The function must have arguments `function(x, eta, extra = NULL, varfun = FALSE)`. The `eta` and `extra` arguments are used to obtain the parameter values. If `varfun = TRUE` then the function returns the variance function, else the characteristic function (default). Note that one should check that the `infos` slot has a list component called `charfun` which is `TRUE` before attempting to use this slot. This is an easier way to test that this slot is operable.

`constraints`: Object of class `"expression"` which sets up any constraint matrices defined by arguments in the family function. A `zero` argument is always fed into `cm.zero.vgam`, whereas other constraints are fed into `cm.vgam`.

`deviance`: Object of class `"function"` returning the deviance of the model. This slot is optional. If present, the function must have arguments `function(mu, y, w, residuals = FALSE, eta, extra = NULL)`. Deviance residuals are returned if `residuals = TRUE`.

`rqresslot`: Object of class `"function"` returning the randomized quantile residuals of the distribution. This slot is optional. If present, the function must have arguments `function(mu, y, w, eta, extra = NULL)`.

`fini1`: Object of class `"expression"` to insert code at a special position in `vglm.fit` or `vgam.fit`. This code is evaluated immediately after the fitting.

`first`: Object of class `"expression"` to insert code at a special position in `vglm` or `vgam`.

`infos`: Object of class `"function"` which returns a list with components such as `M1`. At present only a very few **VGAM** family functions have this feature implemented. Those that do do not require specifying the `M1` argument when used with `rcim`.

initialize: Object of class "expression" used to perform error checking (especially for the variable y) and obtain starting values for the model. In general, etastart or mustart are assigned values based on the variables y, x and w.

linkinv: Object of class "function" which returns the fitted values, given the linear/additive predictors. The function must have arguments function(eta, extra = NULL).

last: Object of class "expression" to insert code at a special position (at the very end) of vglm.fit() or vgam.fit(). This code is evaluated after the fitting. The list misc is often assigned components in this slot, which becomes the misc slot on the fitted object.

linkfun: Object of class "function" which, given the fitted values, returns the linear/additive predictors. If present, the function must have arguments function(mu, extra = NULL). Most **VGAM** family functions do not have a linkfun function. They largely are for classical exponential families, i.e., GLMs.

loglikelihood: Object of class "function" returning the log-likelihood of the model. This slot is optional. If present, the function must have arguments function(mu, y, w, residuals = FALSE, eta, extra = NULL). The argument residuals can be ignored because log-likelihood residuals aren't defined.

middle1: Object of class "expression" to insert code at a special position in vglm.fit or vgam.fit.

middle2: Object of class "expression" to insert code at a special position in vglm.fit or vgam.fit.

simslot: Object of class "function" to allow simulate to work.

hadof: Object of class "function"; experimental.

summary.dispersion: Object of class "logical" indicating whether the general VGLM formula (based on a residual sum of squares) can be used for computing the scaling/dispersion parameter. It is TRUE for most models except for nonlinear regression models.

vfamily: Object of class "character" giving class information about the family function. Although not developed at this stage, more flexible classes are planned in the future. For example, family functions sratio, cratio, cumulative, and acat all operate on categorical data, therefore will have a special class called "VGAMcat", say. Then if fit was a vglm object, then coef(fit) would print out the vglm coefficients plus "VGAMcat" information as well.

deriv: Object of class "expression" which returns a M-column matrix of first derivatives of the log-likelihood function with respect to the linear/additive predictors, i.e., the score vector. In Yee and Wild (1996) this is the $\boldsymbol{d}_i$ vector. Thus each row of the matrix returned by this slot is such a vector.

weight: Object of class "expression" which returns the second derivatives of the log-likelihood function with respect to the linear/additive predictors. This can be either the observed or expected information matrix, i.e., Newton-Raphson or Fisher-scoring respectively. In Yee and Wild (1996) this is the $\boldsymbol{W}_i$ matrix. Thus each row of the matrix returned by this slot is such a matrix. Like the weights slot of vglm/vgam, it is stored in *matrix-band* form, whereby the first M columns of the matrix are the diagonals, followed by the upper-diagonal band, followed by the band above that, etc. In this case, there can be up to $M(M+1)$ columns, with the last column corresponding to the (1,M) elements of the weight matrices.

validfitted, validparams: Functions that test that the fitted values and all parameters are within range. These functions can issue a warning if violations are detected.

Methods

print signature(x = "vglmff"): short summary of the family function.

Warning

VGAM family functions are not compatible with `glm`, nor `gam()` (from either **gam** or **mgcv**).

Note

With link functions etc., one must use `substitute` to embed the options into the code. There are two different forms: `eval(substitute(expression({...}), list(...)))` for expressions, and `eval(substitute( function(...) { ... }, list(...) ))` for functions.

The `extra` argument in `linkinv`, `linkfun`, `deviance`, `loglikelihood`, etc. matches with the argument `extra` in `vglm`, `vgam` and `rrvglm`. This allows input to be fed into all slots of a **VGAM** family function.

The expression `derivative` is evaluated immediately prior to `weight`, so there is provision for re-use of variables etc. Programmers must be careful to choose variable names that do not interfere with `vglm.fit`, `vgam.fit()` etc.

Programmers of **VGAM** family functions are encouraged to keep to previous conventions regarding the naming of arguments, e.g., `link` is the argument for parameter link functions, `zero` for allowing some of the linear/additive predictors to be an intercept term only, etc.

In general, Fisher-scoring is recommended over Newton-Raphson where tractable. Although usually slightly slower in convergence, the weight matrices from using the expected information are positive-definite over a larger parameter space.

Author(s)

Thomas W. Yee

References

Yee, T. W. and Wild, C. J. (1996). Vector generalized additive models. *Journal of the Royal Statistical Society, Series B, Methodological*, **58**, 481–493.

See Also

`vglm`, `vgam`, `rrvglm`, `rcim`.

Examples

```
cratio()
cratio(link = "clogloglink")
cratio(link = "clogloglink", reverse = TRUE)
```

vonmises *von Mises Distribution Family Function*

Description

Estimates the location and scale parameters of the von Mises distribution by maximum likelihood estimation.

Usage

```
vonmises(llocation = extlogitlink(min = 0, max = 2*pi),
         lscale = "loglink", ilocation = NULL, iscale = NULL,
         imethod = 1, zero = NULL)
```

Arguments

`llocation, lscale`
: Parameter link functions applied to the location a parameter and scale parameter k, respectively. See `Links` for more choices. For k, a log link is the default because the parameter is positive.

`ilocation`
: Initial value for the location a parameter. By default, an initial value is chosen internally using `imethod`. Assigning a value will override the argument `imethod`.

`iscale`
: Initial value for the scale k parameter. By default, an initial value is chosen internally using `imethod`. Assigning a value will override the argument `imethod`.

`imethod`
: An integer with value `1` or `2` which specifies the initialization method. If failure to converge occurs try the other value, or else specify a value for `ilocation` and `iscale`.

`zero`
: An integer-valued vector specifying which linear/additive predictors are modelled as intercepts only. The default is none of them. If used, one can choose one value from the set {1,2}. See `CommonVGAMffArguments` for more information.

Details

The (two-parameter) von Mises is the most commonly used distribution in practice for circular data. It has a density that can be written as

$$f(y; a, k) = \frac{\exp[k \cos(y - a)]}{2\pi I_0(k)}$$

where $0 \leq y < 2\pi$, $k > 0$ is the scale parameter, a is the location parameter, and $I_0(k)$ is the modified Bessel function of order 0 evaluated at k. The mean of Y (which is the fitted value) is a and the circular variance is $1 - I_1(k)/I_0(k)$ where $I_1(k)$ is the modified Bessel function of order 1. By default, $\eta_1 = \log(a/(2\pi - a))$ and $\eta_2 = \log(k)$ for this family function.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, rrvglm and vgam.

Warning

Numerically, the von Mises can be difficult to fit because of a log-likelihood having multiple maximums. The user is therefore encouraged to try different starting values, i.e., make use of ilocation and iscale.

Note

The response and the fitted values are scaled so that $0 \leq y < 2\pi$. The linear/additive predictors are left alone. Fisher scoring is used.

Author(s)

T. W. Yee

References

Forbes, C., Evans, M., Hastings, N. and Peacock, B. (2011). *Statistical Distributions*, Hoboken, NJ, USA: John Wiley and Sons, Fourth edition.

See Also

Bessel, cardioid.

CircStats and **circular** currently have a lot more R functions for circular data than the **VGAM** package.

Examples

```
vdata <- data.frame(x2 = runif(nn <- 1000))
vdata <- transform(vdata,
                   y = rnorm(nn, 2+x2, exp(0.2)))  # Bad data!!
fit <- vglm(y  ~ x2, vonmises(zero = 2), vdata, trace = TRUE)
coef(fit, matrix = TRUE)
Coef(fit)
with(vdata, range(y))  # Original data
range(depvar(fit))     # Processed data is in [0,2*pi)
```

`vplot.profile` *Plotting Functions for 'profile' Objects*

Description

`plot` and `pairs` methods for objects of class `"profile"`, but renamed as `vplot` and `vpairs`.

Usage

```
vplot.profile(x, ...)
vpairs.profile(x, colours = 2:3, ...)
```

Arguments

`x`	an object inheriting from class `"profile"`.
`colours`	Colours to be used for the mean curves conditional on `x` and `y` respectively.
`...`	arguments passed to or from other methods.

Details

See `profile.glm` for details.

Author(s)

T. W. Yee adapted this function from `profile.glm`, written originally by D. M. Bates and W. N. Venables. (For S in 1996.)

See Also

`profilevglm`, `confintvglm`, `lrt.stat`, `profile.glm`, `profile.nls`.

Examples

```
pneumo <- transform(pneumo, let = log(exposure.time))
fit1 <- vglm(cbind(normal, mild, severe) ~ let, acat,
             trace = TRUE, data = pneumo)
pfit1 <- profile(fit1, trace = FALSE)
## Not run:
vplot.profile(pfit1)
vpairs.profile(pfit1)

## End(Not run)
```

vsmooth.spline *Vector Cubic Smoothing Spline*

Description

Fits a vector cubic smoothing spline.

Usage

```
vsmooth.spline(x, y, w = NULL, df = rep(5, M), spar = NULL,
               i.constraint = diag(M),
               x.constraint = diag(M),
               constraints = list("(Intercepts)" = i.constraint,
                                  x = x.constraint),
               all.knots = FALSE, var.arg = FALSE, scale.w = TRUE,
               nk = NULL, control.spar = list())
```

Arguments

- `x` A vector, matrix or a list. If a list, the `x` component is used. If a matrix, the first column is used. `x` may also be a complex vector, in which case the real part is used, and the imaginary part is used for the response. In this help file, `n` is the number of unique values of `x`.
- `y` A vector, matrix or a list. If a list, the `y` component is used. If a matrix, all but the first column is used. In this help file, `M` is the number of columns of `y` if there are no constraints on the functions.
- `w` The weight matrices or the number of observations. If the weight matrices, then this must be a `n`-row matrix with the elements in matrix-band form (see `iam`). If a vector, then these are the number of observations. By default, `w` is the `M` by `M` identity matrix, denoted by `matrix(1, n, M)`.
- `df` Numerical vector containing the degrees of freedom for each component function (smooth). If necessary, the vector is recycled to have length equal to the number of component functions to be estimated (`M` if there are no constraints), which equals the number of columns of the `x`-constraint matrix. A value of 2 means a linear fit, and each element of `df` should lie between 2 and `n`. The larger the values of `df` the more wiggly the smooths.
- `spar` Numerical vector containing the non-negative smoothing parameters for each component function (smooth). If necessary, the vector is recycled to have length equal to the number of component functions to be estimated (`M` if there are no constraints), which equals the number of columns of the `x`-constraint matrix. A value of zero means the smooth goes through the data and hence is wiggly. A value of `Inf` may be assigned, meaning the smooth will be linear. By default, the `NULL` value of `spar` means `df` is used to determine the smoothing parameters.
- `all.knots` Logical. If `TRUE` then each distinct value of `x` will be a knot. By default, only a subset of the unique values of `x` are used; typically, the number of knots is `O(n^0.25)` for `n` large, but if `n <= 40` then all the unique values of `x` are used.

- `i.constraint` — A `M`-row constraint matrix for the intercepts. It must be of full column rank. By default, the constraint matrix for the intercepts is the `M` by `M` identity matrix, meaning no constraints.
- `x.constraint` — A `M`-row constraint matrix for `x`. It must be of full column rank. By default, the constraint matrix for the intercepts is the `M` by `M` identity matrix, meaning no constraints.
- `constraints` — An alternative to specifying `i.constraint` and `x.constraint`, this is a list with two components corresponding to the intercept and `x` respectively. They must both be a `M`-row constraint matrix with full column rank.
- `var.arg` — Logical: return the pointwise variances of the fit? Currently, this corresponds only to the nonlinear part of the fit, and may be wrong.
- `scale.w` — Logical. By default, the weights `w` are scaled so that the diagonal elements have mean 1.
- `nk` — Number of knots. If used, this argument overrides `all.knots`, and must lie between 6 and `n`+2 inclusive.
- `control.spar` — See `smooth.spline`.

Details

The algorithm implemented is detailed in Yee (2000). It involves decomposing the component functions into a linear and nonlinear part, and using B-splines. The cost of the computation is `O(n M^3)`.

The argument `spar` contains *scaled* smoothing parameters.

Value

An object of class `"vsmooth.spline"` (see `vsmooth.spline-class`).

WARNING

See `vgam` for information about an important bug.

Note

This function is quite similar to `smooth.spline` but offers less functionality. For example, cross validation is not implemented here. For `M = 1`, the results will be generally different, mainly due to the different way the knots are selected.

The vector cubic smoothing spline which `s()` represents is computationally demanding for large M. The cost is approximately $O(nM^3)$ where n is the number of unique abscissae.

Yet to be done: return the *unscaled* smoothing parameters.

Author(s)

Thomas W. Yee

References

Yee, T. W. (2000). Vector Splines and Other Vector Smoothers. Pages 529–534. In: Bethlehem, J. G. and van der Heijde, P. G. M. *Proceedings in Computational Statistics COMPSTAT 2000*. Heidelberg: Physica-Verlag.

See Also

`vsmooth.spline-class`, `plot.vsmooth.spline`, `predict.vsmooth.spline`, `iam`, `sm.os`, `s`, `smooth.spline`.

Examples

```
nn <- 20; x <- 2 + 5*(nn:1)/nn
x[2:4] <- x[5:7]  # Allow duplication
y1 <- sin(x) + rnorm(nn, sd = 0.13)
y2 <- cos(x) + rnorm(nn, sd = 0.13)
y3 <- 1 + sin(x) + rnorm(nn, sd = 0.13)  # For constraints
y <- cbind(y1, y2, y3)
ww <- cbind(rep(3, nn), 4, (1:nn)/nn)

(fit <- vsmooth.spline(x, y, w = ww, df = 5))
## Not run:
plot(fit)  # The 1st & 3rd functions dont differ by a constant

## End(Not run)

mat <- matrix(c(1,0,1, 0,1,0), 3, 2)
(fit2 <- vsmooth.spline(x, y, w = ww, df = 5, i.constr = mat,
                        x.constr = mat))
# The 1st and 3rd functions do differ by a constant:
mycols <- c("orange", "blue", "orange")
## Not run:  plot(fit2, lcol = mycols, pcol = mycols, las = 1)

p <- predict(fit, x = model.matrix(fit, type = "lm"), deriv = 0)
max(abs(depvar(fit) - with(p, y)))  # Should be 0

par(mfrow = c(3, 1))
ux <- seq(1, 8, len = 100)
for (dd in 1:3) {
  pp <- predict(fit, x = ux, deriv = dd)
## Not run:
with(pp, matplot(x, y, type = "l", main = paste("deriv =", dd),
                 lwd = 2, ylab = "", cex.axis = 1.5,
                 cex.lab = 1.5, cex.main = 1.5))
## End(Not run)
}
```

`waitakere` *Waitakere Ranges Data*

Description

The `waitakere` data frame has 579 rows and 18 columns. Altitude is explanatory, and there are binary responses (presence/absence = 1/0 respectively) for 17 plant species.

Usage

```
data(waitakere)
```

Format

This data frame contains the following columns:

agaaus Agathis australis, or Kauri

beitaw Beilschmiedia tawa, or Tawa

corlae Corynocarpus laevigatus

cyadea Cyathea dealbata

cyamed Cyathea medullaris

daccup Dacrydium cupressinum

dacdac Dacrycarpus dacrydioides

eladen Elaecarpus dentatus

hedarb Hedycarya arborea

hohpop Species name unknown

kniexc Knightia excelsa, or Rewarewa

kuneri Kunzea ericoides

lepsco Leptospermum scoparium

metrob Metrosideros robusta

neslan Nestegis lanceolata

rhosap Rhopalostylis sapida

vitluc Vitex lucens, or Puriri

altitude meters above sea level

Details

These were collected from the Waitakere Ranges, a small forest in northern Auckland, New Zealand. At 579 sites in the forest, the presence/absence of 17 plant species was recorded, as well as the altitude. Each site was of area size $200m^2$.

Source

Dr Neil Mitchell, University of Auckland.

See Also

hunua.

Examples

```
fit <- vgam(agaaus ~ s(altitude, df = 2), binomialff, waitakere)
head(predict(fit, waitakere, type = "response"))
## Not run:  plot(fit, se = TRUE, lcol = "orange", scol = "blue")
```

---

wald.stat *Wald Test Statistics Evaluated at the Null Values*

Description

Generic function that computes Wald test statistics evaluated at the null values (consequently they do not suffer from the Hauck-Donner effect).

Usage

```
wald.stat(object, ...)
wald.stat.vlm(object, values0 = 0, subset = NULL, omit1s = TRUE,
          all.out = FALSE, orig.SE = FALSE, iterate.SE = TRUE,
          trace = FALSE, ...)
```

Arguments

- `object` — A `vglm` fit.
- `values0` — Numeric vector. The null values corresponding to the null hypotheses. Recycled if necessary.
- `subset` — Same as in `hdeff`.
- `omit1s` — Logical. Does one omit the intercepts? Because the default would be to test that each intercept is equal to 0, which often does not make sense or is unimportant, the intercepts are not tested by default. If they are tested then each linear predictor must have at least one coefficient (from another variable) to be estimated.
- `all.out` — Logical. If `TRUE` then a list is returned containing various quantities such as the SEs, instead of just the Wald statistics.
- `orig.SE` — Logical. If `TRUE` then the standard errors are computed at the MLE (of the original object). In practice, the (usual or unmodified) Wald statistics etc. are extracted from `summary(object)` because it was computed there. These may suffer from the HDE since *all* the SEs are evaluated at the MLE of the original object. If `TRUE` then argument `iterate.SE` may be ignored or overwritten. If `orig.SE = FALSE` then the kth SE uses the kth value of `values0` in its computation and `iterate.SE` specifies the choice of the other coefficients.

  This argument was previously called `as.summary` because if `TRUE` then the Wald statistics are the same as `summary(glm())`.

  For one-parameter models setting `orig.SE = FALSE` results in what is called the *null Wald* (NW) statistic by some people, e.g., Laskar and King (1997) and Goh and King (1999). The NW statistic does not suffer from the HDE.

`iterate.SE`	Logical, for the standard error computations. If `TRUE` then IRLS iterations are performed to get MLEs of the *other* regression coefficients, subject to one coefficient being equal to the appropriate `values0` value. If `FALSE` then the other regression coefficients have values obtained at the original fit. It is recommended that a `TRUE` be used as the answer tends to be more accurate. If the large (VLM) model matrix only has one column and `iterate.SE = TRUE` then an error will occur because there are no *other* regression coefficients to estimate.
`trace`	Logical. If `TRUE` then some output is produced as the IRLS iterations proceed. The value `NULL` means to use the `trace` value of the fitted object; see `vglm.control`.
`...`	Ignored for now.

Details

By default, `summaryvglm` and most regression modelling functions such as `summary.glm` compute all the standard errors (SEs) of the estimates at the MLE and not at 0. This corresponds to `orig.SE = TRUE` and it is vulnerable to the Hauck-Donner effect (HDE; see `hdeff`). One solution is to compute the SEs at 0 (or more generally, at the values of the argument `values0`). This function does that. The two variants of Wald statistics are asymptotically equivalent; however in small samples there can be an appreciable difference, and the difference can be large if the estimates are near to the boundary of the parameter space.

None of the tests here are joint, hence the degrees of freedom is always unity. For a factor with more than 2 levels one can use `anova.vglm` to test for the significance of the factor. If `orig.SE = FALSE` and `iterate.SE = FALSE` then one retains the MLEs of the original fit for the values of the other coefficients, and replaces one coefficient at a time by the value 0 (or whatever specified by `values0`). One alternative would be to recompute the MLEs of the other coefficients after replacing one of the values; this is the default because `iterate.SE = TRUE` and `orig.SE = FALSE`. Just like with the original IRLS iterations, the iterations here are not guaranteed to converge.

Almost all **VGAM** family functions use the EIM and not the OIM; this affects the resulting standard errors. Also, regularity conditions are assumed for the Wald, likelihood ratio and score tests; some **VGAM** family functions such as `alaplace1` are experimental and do not satisfy such conditions, therefore naive inference is hazardous.

The default output of this function can be seen by setting `wald0.arg = TRUE` in `summaryvglm`.

Value

By default the signed square root of the Wald statistics whose SEs are computed at one each of the null values. If `all.out = TRUE` then a list is returned with the following components: `wald.stat` the Wald statistic, `SE0` the standard error of that coefficient, `values0` the null values. Approximately, the default Wald statistics output are standard normal random variates if each null hypothesis is true.

Altogether, by the four combinations of `iterate.SE` and `orig.SE`, there are three different variants of the Wald statistic that can be returned.

Warning

This function has been tested but not thoroughly. Convergence failure is possible for some models applied to certain data sets; it is a good idea to set `trace = TRUE` to monitor convergence. For example, for a particular explanatory variable, the estimated regression coefficients of a non-parallel cumulative logit model (see cumulative) are ordered, and perturbing one coefficient might disrupt the order and create numerical problems.

Author(s)

Thomas W. Yee

References

Laskar, M. R. and M. L. King (1997). Modified Wald test for regression disturbances. *Economics Letters*, **56**, 5–11.

Goh, K.-L. and M. L. King (1999). A correction for local biasedness of the Wald and null Wald tests. *Oxford Bulletin of Economics and Statistics* **61**, 435–450.

See Also

lrt.stat, score.stat, summaryvglm, summary.glm, anova.vglm, vglm, hdeff, hdeffsev.

Examples

```
set.seed(1)
pneumo <- transform(pneumo, let = log(exposure.time),
                            x3 = rnorm(nrow(pneumo)))
(fit <- vglm(cbind(normal, mild, severe) ~ let + x3, propodds, pneumo))
wald.stat(fit)  # No HDE here
summary(fit, wald0 = TRUE)  # See them here
coef(summary(fit))  # Usual Wald statistics evaluated at the MLE
wald.stat(fit, orig.SE = TRUE)  # Same as previous line
```

---

waldff *Wald Distribution Family Function*

Description

Estimates the parameter of the standard Wald distribution by maximum likelihood estimation.

Usage

```
waldff(llambda = "loglink", ilambda = NULL)
```

Arguments

llambda,ilambda
: See CommonVGAMffArguments for information.

Details

The standard Wald distribution is a special case of the inverse Gaussian distribution with $\mu = 1$. It has a density that can be written as

$$f(y; \lambda) = \sqrt{\lambda/(2\pi y^3)} \ \exp\left(-\lambda(y-1)^2/(2y)\right)$$

where $y > 0$ and $\lambda > 0$. The mean of Y is 1 (returned as the fitted values) and its variance is $1/\lambda$. By default, $\eta = \log(\lambda)$.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Note

The **VGAM** family function inv.gaussianff estimates the location parameter μ too.

Author(s)

T. W. Yee

References

Johnson, N. L. and Kotz, S. and Balakrishnan, N. (1994). *Continuous Univariate Distributions*, 2nd edition, Volume 1, New York: Wiley.

See Also

inv.gaussianff, rinv.gaussian.

Examples

```
wdata <- data.frame(y = rinv.gaussian(1000, mu =  1, exp(1)))
wfit <- vglm(y ~ 1, waldff(ilambda = 0.2), wdata, trace = TRUE)
coef(wfit, matrix = TRUE)
Coef(wfit)
summary(wfit)
```

---

weibull.mean *Weibull Distribution Family Function, Parameterized by the Mean*

Description

Maximum likelihood estimation of the 2-parameter Weibull distribution. The mean is one of the parameters. No observations should be censored.

Usage

```
weibull.mean(lmean = "loglink", lshape = "loglink",
     imean = NULL, ishape = NULL, probs.y = c(0.2, 0.5, 0.8),
     imethod = 1, zero = "shape")
```

Arguments

`lmean, lshape` Parameter link functions applied to the (positive) mean parameter (called mu below) and (positive) shape parameter (called a below). See `Links` for more choices.

`imean, ishape` Optional initial values for the mean and shape parameters.

`imethod, zero, probs.y` Details at `CommonVGAMffArguments`.

Details

See `weibullR` for most of the details for this family function too. The mean of Y is $b\,\Gamma(1 + 1/a)$ (returned as the fitted values), and this is the first parameter (a `loglink` link is the default because it is positive). The other parameter is the positive shape paramter a, also having a default `loglink` link.

This **VGAM** family function currently does not handle censored data. Fisher scoring is used to estimate the two parameters. Although the expected information matrices used here are valid in all regions of the parameter space, the regularity conditions for maximum likelihood estimation are satisfied only if $a > 2$ (according to Kleiber and Kotz (2003)). If this is violated then a warning message is issued. One can enforce $a > 2$ by choosing `lshape = logofflink(offset = -2)`. Common values of the shape parameter lie between 0.5 and 3.5.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Note

See `weibullR` for more details. This **VGAM** family function handles multiple responses.

Author(s)

T. W. Yee

See Also

`weibullR`, `dweibull`, `truncweibull`, `gev`, `lognormal`, `expexpff`, `maxwell`, `rayleigh`, `gumbelII`.

Examples

```
wdata <- data.frame(x2 = runif(nn <- 1000))  # Complete data
wdata <- transform(wdata, mu     = exp(-1 + 1 * x2),
                          x3     = rnorm(nn),
                          shape1 = exp(1),
                          shape2 = exp(2))
wdata <- transform(wdata,
  y1 = rweibull(nn, shape1, scale = mu / gamma(1 + 1/shape1)),
  y2 = rweibull(nn, shape2, scale = mu / gamma(1 + 1/shape2)))
fit <- vglm(cbind(y1, y2) ~ x2 + x3, weibull.mean, wdata,
            trace = TRUE)
coef(fit, matrix = TRUE)
sqrt(diag(vcov(fit)))  # SEs
summary(fit, presid = FALSE)
```

---

weibullR *Weibull Distribution Family Function*

Description

Maximum likelihood estimation of the 2-parameter Weibull distribution. No observations should be censored.

Usage

```
weibullR(lscale = "loglink", lshape = "loglink",
         iscale = NULL,   ishape = NULL, lss = TRUE, nrfs = 1,
         probs.y = c(0.2, 0.5, 0.8), imethod = 1, zero = "shape")
```

Arguments

`lshape, lscale` Parameter link functions applied to the (positive) shape parameter (called a below) and (positive) scale parameter (called b below). See `Links` for more choices.

`ishape, iscale` Optional initial values for the shape and scale parameters.

`nrfs` Currently this argument is ignored. Numeric, of length one, with value in $[0, 1]$. Weighting factor between Newton-Raphson and Fisher scoring. The value 0 means pure Newton-Raphson, while 1 means pure Fisher scoring. The default value uses a mixture of the two algorithms, and retaining positive-definite working weights.

`imethod` Initialization method used if there are censored observations. Currently only the values 1 and 2 are allowed.

`zero, probs.y, lss` Details at `CommonVGAMffArguments`.

Details

The Weibull density for a response Y is

$$f(y; a, b) = a y^{a-1} \exp[-(y/b)^a]/(b^a)$$

for $a > 0$, $b > 0$, $y > 0$. The cumulative distribution function is

$$F(y; a, b) = 1 - \exp[-(y/b)^a].$$

The mean of Y is $b\,\Gamma(1 + 1/a)$ (returned as the fitted values), and the mode is at $b\,(1 - 1/a)^{1/a}$ when $a > 1$. The density is unbounded for $a < 1$. The kth moment about the origin is $E(Y^k) = b^k\,\Gamma(1 + k/a)$. The hazard function is $a t^{a-1}/b^a$.

This **VGAM** family function currently does not handle censored data. Fisher scoring is used to estimate the two parameters. Although the expected information matrices used here are valid in all regions of the parameter space, the regularity conditions for maximum likelihood estimation are satisfied only if $a > 2$ (according to Kleiber and Kotz (2003)). If this is violated then a warning message is issued. One can enforce $a > 2$ by choosing `lshape = logofflink(offset = -2)`. Common values of the shape parameter lie between 0.5 and 3.5.

Summarized in Harper et al. (2011), for inference, there are 4 cases to consider. If $a \leq 1$ then the MLEs are not consistent (and the smallest observation becomes a hyperefficient solution for the location parameter in the 3-parameter case). If $1 < a < 2$ then MLEs exist but are not asymptotically normal. If $a = 2$ then the MLEs exist and are normal and asymptotically efficient but with a slower convergence rate than when $a > 2$. If $a > 2$ then MLEs have classical asymptotic properties.

The 3-parameter (location is the third parameter) Weibull can be estimated by maximizing a profile log-likelihood (see, e.g., Harper et al. (2011) and Lawless (2003)), else try `gev` which is a better parameterization.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Warning

This function is under development to handle other censoring situations. The version of this function which will handle censored data will be called `cenweibull()`. It is currently being written and will use `SurvS4` as input. It should be released in later versions of **VGAM**.

If the shape parameter is less than two then misleading inference may result, e.g., in the `summary` and `vcov` of the object.

Note

Successful convergence depends on having reasonably good initial values. If the initial values chosen by this function are not good, make use the two initial value arguments.

This **VGAM** family function handles multiple responses.

The Weibull distribution is often an alternative to the lognormal distribution. The inverse Weibull distribution, which is that of $1/Y$ where Y has a Weibull(a, b) distribution, is known as the log-Gompertz distribution.

There are problems implementing the three-parameter Weibull distribution. These are because the classical regularity conditions for the asymptotic properties of the MLEs are not satisfied because the support of the distribution depends on one of the parameters.

Other related distributions are the Maxwell and Rayleigh distributions.

Author(s)

T. W. Yee

References

Kleiber, C. and Kotz, S. (2003). *Statistical Size Distributions in Economics and Actuarial Sciences*, Hoboken, NJ, USA: Wiley-Interscience.

Johnson, N. L. and Kotz, S. and Balakrishnan, N. (1994). *Continuous Univariate Distributions*, 2nd edition, Volume 1, New York: Wiley.

Lawless, J. F. (2003). *Statistical Models and Methods for Lifetime Data*, 2nd ed. Hoboken, NJ, USA: John Wiley & Sons.

Rinne, Horst. (2009). *The Weibull Distribution: A Handbook*. Boca Raton, FL, USA: CRC Press.

Gupta, R. D. and Kundu, D. (2006). On the comparison of Fisher information of the Weibull and GE distributions, *Journal of Statistical Planning and Inference*, **136**, 3130–3144.

Harper, W. V. and Eschenbach, T. G. and James, T. R. (2011). Concerns about Maximum Likelihood Estimation for the Three-Parameter Weibull Distribution: Case Study of Statistical Software, *The American Statistician*, **65(1)**, 44–54.

Smith, R. L. (1985). Maximum likelihood estimation in a class of nonregular cases. *Biometrika*, **72**, 67–90.

Smith, R. L. and Naylor, J. C. (1987). A comparison of maximum likelihood and Bayesian estimators for the three-parameter Weibull distribution. *Applied Statistics*, **36**, 358–369.

See Also

weibull.mean, dweibull, truncweibull, gev, lognormal, expexpff, maxwell, rayleigh, gumbelII.

Examples

```
wdata <- data.frame(x2 = runif(nn <- 1000))  # Complete data
wdata <- transform(wdata,
            y1 = rweibull(nn, exp(1), scale = exp(-2 + x2)),
            y2 = rweibull(nn, exp(2), scale = exp( 1 - x2)))
fit <- vglm(cbind(y1, y2) ~ x2, weibullR, wdata, trace = TRUE)
coef(fit, matrix = TRUE)
vcov(fit)
summary(fit)
```

weightsvglm *Prior and Working Weights of a VGLM fit*

Description

Returns either the prior weights or working weights of a VGLM object.

Usage

```
weightsvglm(object, type = c("prior", "working"),
            matrix.arg = TRUE, ignore.slot = FALSE,
            deriv.arg = FALSE, ...)
```

Arguments

`object`	a model object from the **VGAM** R package that inherits from a *vector generalized linear model* (VGLM), e.g., a model of class `"vglm"`.
`type`	Character, which type of weight is to be returned? The default is the first one.
`matrix.arg`	Logical, whether the answer is returned as a matrix. If not, it will be a vector.
`ignore.slot`	Logical. If `TRUE` then `object@weights` is ignored even if it has been assigned, and the long calculation for `object@weights` is repeated. This may give a slightly different answer because of the final IRLS step at convergence may or may not assign the latest value of quantities such as the mean and weights.
`deriv.arg`	Logical. If `TRUE` then a list with components `deriv` and `weights` is returned. See below for more details.
`...`	Currently ignored.

Details

Prior weights are usually inputted with the `weights` argument in functions such as `vglm` and `vgam`. It may refer to frequencies of the individual data or be weight matrices specified beforehand.

Working weights are used by the IRLS algorithm. They correspond to the second derivatives of the log-likelihood function with respect to the linear predictors. The working weights correspond to positive-definite weight matrices and are returned in *matrix-band* form, e.g., the first M columns correspond to the diagonals, etc.

If one wants to perturb the linear predictors then the `fitted.values` slots should be assigned to the object before calling this function. The reason is that, for some family functions, the variable `mu` is used directly as one of the parameter estimates, without recomputing it from `eta`.

Value

If `type = "working"` and `deriv = TRUE` then a list is returned with the two components described below. Otherwise the prior or working weights are returned depending on the value of `type`.

deriv
: Typically the first derivative of the log-likelihood with respect to the linear predictors. For example, this is the variable `deriv.mu` in `vglm.fit()`, or equivalently, the matrix returned in the `"deriv"` slot of a **VGAM** family function.

weights
: The working weights.

Note

This function is intended to be similar to `weights.glm` (see `glm`).

Author(s)

Thomas W. Yee

See Also

`glm`, `vglmff-class`, `vglm`.

Examples

```
pneumo <- transform(pneumo, let = log(exposure.time))
(fit <- vglm(cbind(normal, mild, severe) ~ let,
             cumulative(parallel = TRUE, reverse = TRUE), pneumo))
depvar(fit)  # These are sample proportions
weights(fit, type = "prior", matrix = FALSE)  # No. of observations

# Look at the working residuals
nn <- nrow(model.matrix(fit, type = "lm"))
M <- ncol(predict(fit))

wwt <- weights(fit, type="working", deriv=TRUE)  # Matrix-band format
wz <- m2a(wwt$weights, M = M)  # In array format
wzinv <- array(apply(wz, 3, solve), c(M, M, nn))
wresid <- matrix(NA, nn, M)  # Working residuals
for (ii in 1:nn)
  wresid[ii, ] <- wzinv[, , ii, drop = TRUE] %*% wwt$deriv[ii, ]
max(abs(c(resid(fit, type = "work")) - c(wresid)))  # Should be 0

(zedd <- predict(fit) + wresid)  # Adjusted dependent vector
```

`wine` *Bitterness in Wine Data*

Description

This oenological data frame concerns the amount of bitterness in 78 bottles of white wine.

Usage

```
data(wine)
```

Format

A data frame with 4 rows on the following 7 variables.

temp temperature, with levels cold and warm.

contact whether contact of the juice with the skin was allowed or avoided, for a specified period. Two levels: no or yes.

bitter1, bitter2, bitter3, bitter4, bitter5 numeric vectors, the counts. The order is none to most intense.

Details

The data set comes from Randall (1989) and concerns a factorial experiment for investigating factors that affect the bitterness of white wines. There are two factors in the experiment: temperature at the time of crushing the grapes and contact of the juice with the skin. Two bottles of wine were fermented for each of the treatment combinations. A panel of 9 judges were selected and trained for the ability to detect bitterness. Thus there were 72 bottles in total. Originally, the bitterness of the wine were taken on a continuous scale in the interval from 0 (none) to 100 (intense) but later they were grouped using equal lengths into five ordered categories 1, 2, 3, 4 and 5.

Source

Christensen, R. H. B. (2013) Analysis of ordinal data with cumulative link models—estimation with the R-package **ordinal**. R Package Version 2013.9-30. `https://CRAN.R-project.org/package=ordinal`.

Randall, J. H. (1989). The analysis of sensory data by generalized linear model. *Biometrical Journal* **31**(7), 781–793.

Kosmidis, I. (2014). Improved estimation in cumulative link models. *Journal of the Royal Statistical Society, Series B, Methodological*, **76**(1): 169–196.

Examples

```
wine
summary(wine)
```

`wrapup.smart` *Cleans Up After Smart Prediction*

Description

`wrapup.smart` deletes any variables used by smart prediction. Needed by both the modelling function and the prediction function.

Usage

```
wrapup.smart()
```

Details

The variables to be deleted are .smart.prediction, .smart.prediction.counter, and .smart.prediction.mode. The function wrapup.smart is useful in R because these variables are held in smartpredenv.

See Also

setup.smart.

Examples

```
## Not run: # Place this inside modelling functions such as lm, glm, vglm.
wrapup.smart()  # Put at the end of lm

## End(Not run)
```

---

yeo.johnson   *Yeo-Johnson Transformation*

Description

Computes the Yeo-Johnson transformation, which is a normalizing transformation.

Usage

```
yeo.johnson(y, lambda, derivative = 0,
            epsilon = sqrt(.Machine$double.eps), inverse = FALSE)
```

Arguments

y	Numeric, a vector or matrix.
lambda	Numeric. It is recycled to the same length as y if necessary.
derivative	Non-negative integer. The default is the ordinary function evaluation, otherwise the derivative with respect to lambda.
epsilon	Numeric and positive value. The tolerance given to values of lambda when comparing it to 0 or 2.
inverse	Logical. Return the inverse transformation?

Details

The Yeo-Johnson transformation can be thought of as an extension of the Box-Cox transformation. It handles both positive and negative values, whereas the Box-Cox transformation only handles positive values. Both can be used to transform the data so as to improve normality. They can be used to perform LMS quantile regression.

Value

The Yeo-Johnson transformation or its inverse, or its derivatives with respect to lambda, of y.

Note

If `inverse = TRUE` then the argument `derivative = 0` is required.

Author(s)

Thomas W. Yee

References

Yeo, I.-K. and Johnson, R. A. (2000). A new family of power transformations to improve normality or symmetry. *Biometrika*, **87**, 954–959.

Yee, T. W. (2004). Quantile regression via vector generalized additive models. *Statistics in Medicine*, **23**, 2295–2315.

See Also

`lms.yjn`, `boxcox`.

Examples

```
y <- seq(-4, 4, len = (nn <- 200))
ltry <- c(0, 0.5, 1, 1.5, 2)  # Try these values of lambda
lltry <- length(ltry)
psi <- matrix(NA_real_, nn, lltry)
for (ii in 1:lltry)
  psi[, ii] <- yeo.johnson(y, lambda = ltry[ii])

## Not run:
matplot(y, psi, type = "l", ylim = c(-4, 4), lwd = 2,
        lty = 1:lltry, col = 1:lltry, las = 1,
        ylab = "Yeo-Johnson transformation",
        main = "Yeo-Johnson transformation with some lambda values")
abline(v = 0, h = 0)
legend(x = 1, y = -0.5, lty = 1:lltry, legend = as.character(ltry),
       lwd = 2, col = 1:lltry)
## End(Not run)
```

`Yules` *Yule-Simon Distribution*

Description

Density, distribution function, quantile function and random generation for the Yule-Simon distribution.

Usage

```
dyules(x, shape, log = FALSE)
pyules(q, shape, lower.tail = TRUE, log.p = FALSE)
qyules(p, shape)
ryules(n, shape)
```

Arguments

x, q, p, n	Same meaning as in Normal.
shape	See yulesimon.
log, lower.tail, log.p	Same meaning as in pnorm or qnorm.

Details

See yulesimon, the **VGAM** family function for estimating the parameter, for the formula of the probability density function and other details.

Value

dyules gives the density, pyules gives the distribution function, qyules gives the quantile function, and ryules generates random deviates.

Note

Numerical problems may occur with qyules() when p is very close to 1.

Author(s)

T. W. Yee

See Also

yulesimon.

Examples

```
dyules(1:20, 2.1)
ryules(20, 2.1)

round(1000 * dyules(1:8, 2))
table(ryules(1000, 2))

## Not run:  x <- 0:6
plot(x, dyules(x, shape = 2.2), type = "h", las = 1, col = "blue")

## End(Not run)
```

yulesimon *Yule-Simon Family Function*

Description

Estimating the shape parameter of the Yule-Simon distribution.

Usage

```
yulesimon(lshape = "loglink", ishape = NULL, nsimEIM = 200,
          zero = NULL)
```

Arguments

lshape	Link function for the shape parameter, called ρ below. See Links for more choices and for general information.
ishape	Optional initial value for the (positive) parameter. See CommonVGAMffArguments for more information. The default is to obtain an initial value internally. Use this argument if the default fails.
nsimEIM, zero	See CommonVGAMffArguments for more information.

Details

The probability function is
$$f(y; \rho) = \rho * beta(y, \rho + 1),$$
where the parameter $\rho > 0$, $beta$ is the beta function, and $y = 1, 2, \ldots$. The function dyules computes this probability function. The mean of Y, which is returned as fitted values, is $\rho/(\rho - 1)$ provided $\rho > 1$. The variance of Y is $\rho^2/((\rho - 1)^2(\rho - 2))$ provided $\rho > 2$.

The distribution was named after Udny Yule and Herbert A. Simon. Simon originally called it the Yule distribution. This family function can handle multiple responses.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm and vgam.

Author(s)

T. W. Yee

References

Simon, H. A. (1955). On a class of skew distribution functions. *Biometrika*, **42**, 425–440.

See Also

ryules, simulate.vlm.

Examples

```
ydata <- data.frame(x2 = runif(nn <- 1000))
ydata <- transform(ydata, y = ryules(nn, shape = exp(1.5 - x2)))
with(ydata, table(y))
fit <- vglm(y ~ x2, yulesimon, data = ydata, trace = TRUE)
coef(fit, matrix = TRUE)
summary(fit)
```

Zabinom *Zero-Altered Binomial Distribution*

Description

Density, distribution function, quantile function and random generation for the zero-altered binomial distribution with parameter pobs0.

Usage

```
dzabinom(x, size, prob, pobs0 = 0, log = FALSE)
pzabinom(q, size, prob, pobs0 = 0)
qzabinom(p, size, prob, pobs0 = 0)
rzabinom(n, size, prob, pobs0 = 0)
```

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
n	number of observations. If length(n) > 1 then the length is taken to be the number required.
size, prob, log	Parameters from the ordinary binomial distribution (see dbinom).
pobs0	Probability of (an observed) zero, called $pobs0$. The default value of pobs0 = 0 corresponds to the response having a positive binomial distribution.

Details

The probability function of Y is 0 with probability pobs0, else a positive binomial(size, prob) distribution.

Value

dzabinom gives the density and pzabinom gives the distribution function, qzabinom gives the quantile function, and rzabinom generates random deviates.

Note

The argument pobs0 is recycled to the required length, and must have values which lie in the interval $[0, 1]$.

Author(s)

T. W. Yee

See Also

zibinomial, Gaitdbinom.

Examples

```
size <- 10; prob <- 0.15; pobs0 <- 0.05; x <- (-1):7
dzabinom(x, size = size, prob = prob, pobs0 = pobs0)
table(rzabinom(100, size = size, prob = prob, pobs0 = pobs0))

## Not run:  x <- 0:10
barplot(rbind(dzabinom(x, size = size, prob = prob, pobs0 = pobs0),
                dbinom(x, size = size, prob = prob)),
  beside = TRUE, col = c("blue", "orange"), cex.main = 0.7, las = 1,
  ylab = "Probability", names.arg = as.character(x),
  main = paste("ZAB(size = ", size, ", prob = ", prob, ", pobs0 = ", pobs0,
               ") [blue] vs",  " Binom(size = ", size, ", prob = ", prob,
               ") [orange] densities", sep = ""))
## End(Not run)
```

---

zabinomial *Zero-Altered Binomial Distribution*

Description

Fits a zero-altered binomial distribution based on a conditional model involving a Bernoulli distribution and a positive-binomial distribution.

Usage

```
zabinomial(lpobs0 = "logitlink", lprob = "logitlink",
     type.fitted = c("mean", "prob", "pobs0"),
     ipobs0 = NULL, iprob = NULL, imethod = 1, zero = NULL)
zabinomialff(lprob = "logitlink", lonempobs0 = "logitlink",
     type.fitted = c("mean", "prob", "pobs0", "onempobs0"),
     iprob = NULL, ionempobs0 = NULL, imethod = 1, zero = "onempobs0")
```

Arguments

lprob	Parameter link function applied to the probability parameter of the binomial distribution. See Links for more choices.
lpobs0	Link function for the parameter p_0, called pobs0 here. See Links for more choices.
type.fitted	See CommonVGAMffArguments and fittedvlm for information.

`iprob, ipobs0` See `CommonVGAMffArguments`.

`lonempobs0, ionempobs0`
: Corresponding argument for the other parameterization. See details below.

`imethod, zero` See `CommonVGAMffArguments`.

Details

The response Y is zero with probability p_0, else Y has a positive-binomial distribution with probability $1 - p_0$. Thus $0 < p_0 < 1$, which may be modelled as a function of the covariates. The zero-altered binomial distribution differs from the zero-inflated binomial distribution in that the former has zeros coming from one source, whereas the latter has zeros coming from the binomial distribution too. The zero-inflated binomial distribution is implemented in `zibinomial`. Some people call the zero-altered binomial a *hurdle* model.

The input is currently a vector or one-column matrix. By default, the two linear/additive predictors for `zabinomial()` are $(logit(p_0), \log(p))^T$.

The **VGAM** family function `zabinomialff()` has a few changes compared to `zabinomial()`. These are: (i) the order of the linear/additive predictors is switched so the binomial probability comes first; (ii) argument `onempobs0` is now 1 minus the probability of an observed 0, i.e., the probability of the positive binomial distribution, i.e., `onempobs0` is `1-pobs0`; (iii) argument `zero` has a new default so that the `onempobs0` is intercept-only by default. Now `zabinomialff()` is generally recommended over `zabinomial()`. Both functions implement Fisher scoring and neither can handle multiple responses.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

The `fitted.values` slot of the fitted object, which should be extracted by the generic function `fitted`, returns the mean μ (default) which is given by

$$\mu = (1 - p_0)\mu_b/[1 - (1 - \mu_b)^N]$$

where μ_b is the usual binomial mean. If `type.fitted = "pobs0"` then p_0 is returned.

Note

The response should be a two-column matrix of counts, with first column giving the number of successes.

Note this family function allows p_0 to be modelled as functions of the covariates by having `zero = NULL`. It is a conditional model, not a mixture model.

These family functions effectively combine `posbinomial` and `binomialff` into one family function.

Author(s)

T. W. Yee

See Also

dzabinom, zibinomial, posbinomial, spikeplot, binomialff, dbinom, CommonVGAMffArguments.

Examples

```
zdata <- data.frame(x2 = runif(nn <- 1000))
zdata <- transform(zdata, size  = 10,
                          prob  = logitlink(-2 + 3*x2, inverse = TRUE),
                          pobs0 = logitlink(-1 + 2*x2, inverse = TRUE))
zdata <- transform(zdata,
                   y1 = rzabinom(nn, size = size, prob = prob, pobs0 = pobs0))
with(zdata, table(y1))

zfit <- vglm(cbind(y1, size - y1) ~ x2, zabinomial(zero = NULL),
             data = zdata, trace = TRUE)
coef(zfit, matrix = TRUE)
head(fitted(zfit))
head(predict(zfit))
summary(zfit)
```

Zageom   *Zero-Altered Geometric Distribution*

Description

Density, distribution function, quantile function and random generation for the zero-altered geometric distribution with parameter pobs0.

Usage

```
dzageom(x, prob, pobs0 = 0, log = FALSE)
pzageom(q, prob, pobs0 = 0)
qzageom(p, prob, pobs0 = 0)
rzageom(n, prob, pobs0 = 0)
```

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
n	number of observations. If length(n) > 1 then the length is taken to be the number required.
prob, log	Parameters from the ordinary geometric distribution (see dgeom).
pobs0	Probability of (an observed) zero, called $pobs0$. The default value of pobs0 = 0 corresponds to the response having a positive geometric distribution.

Details

The probability function of Y is 0 with probability `pobs0`, else a positive geometric(prob) distribution.

Value

`dzageom` gives the density and `pzageom` gives the distribution function, `qzageom` gives the quantile function, and `rzageom` generates random deviates.

Note

The argument `pobs0` is recycled to the required length, and must have values which lie in the interval $[0, 1]$.

Author(s)

T. W. Yee

See Also

`zageometric`, `zigeometric`, `rposgeom`.

Examples

```
prob <- 0.35; pobs0 <- 0.05; x <- (-1):7
dzageom(x, prob = prob, pobs0 = pobs0)
table(rzageom(100, prob = prob, pobs0 = pobs0))

## Not run:  x <- 0:10
barplot(rbind(dzageom(x, prob = prob, pobs0 = pobs0),
                dgeom(x, prob = prob)), las = 1,
        beside = TRUE, col = c("blue", "orange"), cex.main = 0.7,
        ylab = "Probability", names.arg = as.character(x),
        main = paste("ZAG(prob = ", prob, ", pobs0 = ", pobs0,
                   ") [blue] vs",  " Geometric(prob = ", prob,
                   ") [orange] densities", sep = ""))
## End(Not run)
```

`zageometric` *Zero-Altered Geometric Distribution*

Description

Fits a zero-altered geometric distribution based on a conditional model involving a Bernoulli distribution and a positive-geometric distribution.

Usage

```
zageometric(lpobs0 = "logitlink", lprob = "logitlink",
     type.fitted = c("mean", "prob", "pobs0", "onempobs0"),
     imethod = 1, ipobs0 = NULL, iprob = NULL, zero = NULL)
zageometricff(lprob = "logitlink", lonempobs0 = "logitlink",
     type.fitted = c("mean", "prob", "pobs0", "onempobs0"),
     imethod = 1, iprob = NULL, ionempobs0 = NULL, zero = "onempobs0")
```

Arguments

- `lpobs0`: Link function for the parameter p_0 or ϕ, called `pobs0` or `phi` here. See `Links` for more choices.
- `lprob`: Parameter link function applied to the probability of success, called `prob` or p. See `Links` for more choices.
- `type.fitted`: See `CommonVGAMffArguments` and `fittedvlm` for information.
- `ipobs0, iprob`: Optional initial values for the parameters. If given, they must be in range. For multi-column responses, these are recycled sideways.
- `lonempobs0, ionempobs0`: Corresponding argument for the other parameterization. See details below.
- `zero, imethod`: See `CommonVGAMffArguments`.

Details

The response Y is zero with probability p_0, or Y has a positive-geometric distribution with probability $1 - p_0$. Thus $0 < p_0 < 1$, which is modelled as a function of the covariates. The zero-altered geometric distribution differs from the zero-inflated geometric distribution in that the former has zeros coming from one source, whereas the latter has zeros coming from the geometric distribution too. The zero-inflated geometric distribution is implemented in the **VGAM** package. Some people call the zero-altered geometric a *hurdle* model.

The input can be a matrix (multiple responses). By default, the two linear/additive predictors of `zageometric` are $(logit(\phi), logit(p))^T$.

The **VGAM** family function `zageometricff()` has a few changes compared to `zageometric()`. These are: (i) the order of the linear/additive predictors is switched so the geometric probability comes first; (ii) argument `onempobs0` is now 1 minus the probability of an observed 0, i.e., the probability of the positive geometric distribution, i.e., `onempobs0` is `1-pobs0`; (iii) argument `zero` has a new default so that the `pobs0` is intercept-only by default. Now `zageometricff()` is generally recommended over `zageometric()`. Both functions implement Fisher scoring and can handle multiple responses.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

The `fitted.values` slot of the fitted object, which should be extracted by the generic function `fitted`, returns the mean μ (default) which is given by

$$\mu = (1 - \phi)/p.$$

If type.fitted = "pobs0" then p_0 is returned.

Warning

Convergence for this **VGAM** family function seems to depend quite strongly on providing good initial values.

Inference obtained from summary.vglm and summary.vgam may or may not be correct. In particular, the p-values, standard errors and degrees of freedom may need adjustment. Use simulation on artificial data to check that these are reasonable.

Note

Note this family function allows p_0 to be modelled as functions of the covariates. It is a conditional model, not a mixture model.

This family function effectively combines binomialff and posgeometric() and geometric into one family function. However, posgeometric() is not written because it is trivially related to geometric.

Author(s)

T. W. Yee

See Also

dzageom, geometric, zigeometric, spikeplot, dgeom, CommonVGAMffArguments, simulate.vlm.

Examples

```
zdata <- data.frame(x2 = runif(nn <- 1000))
zdata <- transform(zdata, pobs0 = logitlink(-1 + 2*x2, inverse = TRUE),
                          prob  = logitlink(-2 + 3*x2, inverse = TRUE))
zdata <- transform(zdata, y1 = rzageom(nn, prob = prob, pobs0 = pobs0),
                          y2 = rzageom(nn, prob = prob, pobs0 = pobs0))
with(zdata, table(y1))

fit <- vglm(cbind(y1, y2) ~ x2, zageometric, data = zdata, trace = TRUE)
coef(fit, matrix = TRUE)
head(fitted(fit))
head(predict(fit))
summary(fit)
```

Zanegbin *Zero-Altered Negative Binomial Distribution*

Description

Density, distribution function, quantile function and random generation for the zero-altered negative binomial distribution with parameter pobs0.

Usage

```
dzanegbin(x, size, munb, pobs0 = 0, log = FALSE)
pzanegbin(q, size, munb, pobs0 = 0)
qzanegbin(p, size, munb, pobs0 = 0)
rzanegbin(n, size, munb, pobs0 = 0)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. If `length(n) > 1` then the length is taken to be the number required.
`size, munb, log`	Parameters from the ordinary negative binomial distribution (see `dnbinom`). Some arguments have been renamed slightly.
`pobs0`	Probability of zero, called $pobs0$. The default value of `pobs0 = 0` corresponds to the response having a positive negative binomial distribution.

Details

The probability function of Y is 0 with probability `pobs0`, else a positive negative binomial(μ_{nb}, size) distribution.

Value

`dzanegbin` gives the density and `pzanegbin` gives the distribution function, `qzanegbin` gives the quantile function, and `rzanegbin` generates random deviates.

Note

The argument `pobs0` is recycled to the required length, and must have values which lie in the interval $[0, 1]$.

Author(s)

T. W. Yee

See Also

`Gaitdnbinom`, `zanegbinomial`.

Examples

```
munb <- 3; size <- 4; pobs0 <- 0.3; x <- (-1):7
dzanegbin(x, munb = munb, size = size, pobs0 = pobs0)
table(rzanegbin(100, munb = munb, size = size, pobs0 = pobs0))

## Not run:  x <- 0:10
barplot(rbind(dzanegbin(x, munb = munb, size = size, pobs0 = pobs0),
                dnbinom(x, mu   = munb, size = size)),
```

```
        beside = TRUE, col = c("blue", "green"), cex.main = 0.7,
        ylab = "Probability", names.arg = as.character(x), las = 1,
        main = paste0("ZANB(munb = ", munb, ", size = ", size,",
               pobs0 = ", pobs0,
               ") [blue] vs",  " NB(mu = ", munb, ", size = ", size,
               ") [green] densities"))
## End(Not run)
```

`zanegbinomial` *Zero-Altered Negative Binomial Distribution*

Description

Fits a zero-altered negative binomial distribution based on a conditional model involving a binomial distribution and a positive-negative binomial distribution.

Usage

```
zanegbinomial(zero = "size", type.fitted = c("mean", "munb", "pobs0"),
              mds.min = 1e-3, nsimEIM = 500, cutoff.prob = 0.999,
              eps.trig = 1e-7, max.support = 4000, max.chunk.MB = 30,
              lpobs0 = "logitlink", lmunb = "loglink", lsize = "loglink",
              imethod = 1, ipobs0 = NULL,
              imunb = NULL, iprobs.y = NULL, gprobs.y = (0:9)/10,
              isize = NULL, gsize.mux = exp(c(-30, -20, -15, -10, -6:3)))
zanegbinomialff(lmunb = "loglink", lsize = "loglink", lonempobs0 = "logitlink",
                type.fitted = c("mean", "munb", "pobs0", "onempobs0"),
                isize = NULL, ionempobs0 = NULL, zero = c("size",
              "onempobs0"), mds.min = 1e-3, iprobs.y = NULL, gprobs.y = (0:9)/10,
                cutoff.prob = 0.999, eps.trig = 1e-7, max.support = 4000,
               max.chunk.MB = 30, gsize.mux = exp(c(-30, -20, -15, -10, -6:3)),
                imethod = 1, imunb = NULL,
                nsimEIM = 500)
```

Arguments

`lpobs0` Link function for the parameter p_0, called `pobs0` here. See `Links` for more choices.

`lmunb` Link function applied to the `munb` parameter, which is the mean μ_{nb} of an ordinary negative binomial distribution. See `Links` for more choices.

`lsize` Parameter link function applied to the reciprocal of the dispersion parameter, called `k`. That is, as `k` increases, the variance of the response decreases. See `Links` for more choices.

`type.fitted` See `CommonVGAMffArguments` and `fittedvlm` for information.

`lonempobs0, ionempobs0`
Corresponding argument for the other parameterization. See details below.

ipobs0, imunb, isize
: Optional initial values for p_0 and munb and k. If given then it is okay to give one value for each response/species by inputting a vector whose length is the number of columns of the response matrix.

zero
: Specifies which of the three linear predictors are modelled as intercept-only. All parameters can be modelled as a function of the explanatory variables by setting zero = NULL (not recommended). A negative value means that the value is recycled, e.g., setting -3 means all k are intercept-only for zanegbinomial. See CommonVGAMffArguments for more information.

nsimEIM, imethod
: See CommonVGAMffArguments.

iprobs.y, gsize.mux, gprobs.y
: See negbinomial.

cutoff.prob, eps.trig
: See negbinomial.

mds.min, max.support, max.chunk.MB
: See negbinomial.

Details

The response Y is zero with probability p_0, or Y has a positive-negative binomial distribution with probability $1 - p_0$. Thus $0 < p_0 < 1$, which is modelled as a function of the covariates. The zero-altered negative binomial distribution differs from the zero-inflated negative binomial distribution in that the former has zeros coming from one source, whereas the latter has zeros coming from the negative binomial distribution too. The zero-inflated negative binomial distribution is implemented in the **VGAM** package. Some people call the zero-altered negative binomial a *hurdle* model.

For one response/species, by default, the three linear/additive predictors for zanegbinomial() are $(logit(p_0), \log(\mu_{nb}), \log(k))^T$. This vector is recycled for multiple species.

The **VGAM** family function zanegbinomialff() has a few changes compared to zanegbinomial(). These are: (i) the order of the linear/additive predictors is switched so the negative binomial mean comes first; (ii) argument onempobs0 is now 1 minus the probability of an observed 0, i.e., the probability of the positive negative binomial distribution, i.e., onempobs0 is 1-pobs0; (iii) argument zero has a new default so that the pobs0 is intercept-only by default. Now zanegbinomialff() is generally recommended over zanegbinomial(). Both functions implement Fisher scoring and can handle multiple responses.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

The fitted.values slot of the fitted object, which should be extracted by the generic function fitted, returns the mean μ (default) which is given by

$$\mu = (1 - p_0)\mu_{nb}/[1 - (k/(k + \mu_{nb}))^k].$$

If type.fitted = "pobs0" then p_0 is returned.

Warning

This family function is fragile; it inherits the same difficulties as `posnegbinomial`. Convergence for this **VGAM** family function seems to depend quite strongly on providing good initial values.

This **VGAM** family function is computationally expensive and usually runs slowly; setting `trace = TRUE` is useful for monitoring convergence.

Inference obtained from `summary.vglm` and `summary.vgam` may or may not be correct. In particular, the p-values, standard errors and degrees of freedom may need adjustment. Use simulation on artificial data to check that these are reasonable.

Note

Note this family function allows p_0 to be modelled as functions of the covariates provided `zero` is set correctly. It is a conditional model, not a mixture model. Simulated Fisher scoring is the algorithm.

This family function effectively combines `posnegbinomial` and `binomialff` into one family function.

This family function can handle multiple responses, e.g., more than one species.

Author(s)

T. W. Yee

References

Welsh, A. H., Cunningham, R. B., Donnelly, C. F. and Lindenmayer, D. B. (1996). Modelling the abundances of rare species: statistical models for counts with extra zeros. *Ecological Modelling*, **88**, 297–308.

Yee, T. W. (2014). Reduced-rank vector generalized linear models with two linear predictors. *Computational Statistics and Data Analysis*, **71**, 889–902.

See Also

`gaitdnbinomial`, `posnegbinomial`, `Gaitdnbinom`, `negbinomial`, `binomialff`, `zinegbinomial`, `zipoisson`, `spikeplot`, `dnbinom`, `CommonVGAMffArguments`, `simulate.vlm`.

Examples

```
## Not run:
zdata <- data.frame(x2 = runif(nn <- 2000))
zdata <- transform(zdata, pobs0 = logitlink(-1 + 2*x2, inverse = TRUE))
zdata <- transform(zdata,
         y1 = rzanegbin(nn, munb = exp(0+2*x2), size = exp(1), pobs0 = pobs0),
         y2 = rzanegbin(nn, munb = exp(1+2*x2), size = exp(1), pobs0 = pobs0))
with(zdata, table(y1))
with(zdata, table(y2))

fit <- vglm(cbind(y1, y2) ~ x2, zanegbinomial, data = zdata, trace = TRUE)
coef(fit, matrix = TRUE)
```

```
head(fitted(fit))
head(predict(fit))

## End(Not run)
```

Zapois *Zero-Altered Poisson Distribution*

Description

Density, distribution function, quantile function and random generation for the zero-altered Poisson distribution with parameter `pobs0`.

Usage

```
dzapois(x, lambda, pobs0 = 0, log = FALSE)
pzapois(q, lambda, pobs0 = 0)
qzapois(p, lambda, pobs0 = 0)
rzapois(n, lambda, pobs0 = 0)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. If `length(n) > 1` then the length is taken to be the number required.
`lambda`	Vector of positive means.
`pobs0`	Probability of zero, called $pobs0$. The default value of `pobs0 = 0` corresponds to the response having a positive Poisson distribution.
`log`	Logical. Return the logarithm of the answer?

Details

The probability function of Y is 0 with probability `pobs0`, else a positive $Poisson(\lambda)$.

Value

`dzapois` gives the density, `pzapois` gives the distribution function, `qzapois` gives the quantile function, and `rzapois` generates random deviates.

Note

The argument `pobs0` is recycled to the required length, and must have values which lie in the interval $[0, 1]$.

Author(s)

T. W. Yee

See Also

zapoisson, Gaitdpois, dzipois.

Examples

```
lambda <- 3; pobs0 <- 0.2; x <- (-1):7
(ii <- dzapois(x, lambda, pobs0))
max(abs(cumsum(ii) - pzapois(x, lambda, pobs0)))  # Should be 0
table(rzapois(100, lambda, pobs0))
table(qzapois(runif(100), lambda, pobs0))
round(dzapois(0:10, lambda, pobs0) * 100)  # Should be similar

## Not run:  x <- 0:10
barplot(rbind(dzapois(x, lambda, pobs0), dpois(x, lambda)),
        beside = TRUE, col = c("blue", "green"), las = 1,
        main = paste0("ZAP(", lambda, ", pobs0 = ", pobs0, ") [blue]",
                      "vs Poisson(", lambda, ") [green] densities"),
        names.arg = as.character(x), ylab = "Probability")
## End(Not run)
```

---

zapoisson *Zero-Altered Poisson Distribution*

Description

Fits a zero-altered Poisson distribution based on a conditional model involving a Bernoulli distribution and a positive-Poisson distribution.

Usage

```
zapoisson(lpobs0 = "logitlink", llambda = "loglink", type.fitted =
    c("mean", "lambda", "pobs0", "onempobs0"), imethod = 1,
    ipobs0 = NULL, ilambda = NULL, ishrinkage = 0.95, probs.y = 0.35,
    zero = NULL)
zapoissonff(llambda = "loglink", lonempobs0 = "logitlink", type.fitted =
    c("mean", "lambda", "pobs0", "onempobs0"), imethod = 1,
    ilambda = NULL, ionempobs0 = NULL, ishrinkage = 0.95,
    probs.y = 0.35, zero = "onempobs0")
```

Arguments

lpobs0	Link function for the parameter p_0, called pobs0 here. See Links for more choices.
llambda	Link function for the usual λ parameter. See Links for more choices.

`type.fitted` See CommonVGAMffArguments and fittedvlm for information.

`lonempobs0` Corresponding argument for the other parameterization. See details below.

`imethod, ipobs0, ionempobs0, ilambda, ishrinkage`
See CommonVGAMffArguments for information.

`probs.y, zero` See CommonVGAMffArguments for information.

Details

The response Y is zero with probability p_0, else Y has a positive-Poisson(λ) distribution with probability $1 - p_0$. Thus $0 < p_0 < 1$, which is modelled as a function of the covariates. The zero-altered Poisson distribution differs from the zero-inflated Poisson distribution in that the former has zeros coming from one source, whereas the latter has zeros coming from the Poisson distribution too. Some people call the zero-altered Poisson a *hurdle* model.

For one response/species, by default, the two linear/additive predictors for `zapoisson()` are $(logit(p_0), \log(\lambda))^T$.

The **VGAM** family function `zapoissonff()` has a few changes compared to `zapoisson()`. These are: (i) the order of the linear/additive predictors is switched so the Poisson mean comes first; (ii) argument `onempobs0` is now 1 minus the probability of an observed 0, i.e., the probability of the positive Poisson distribution, i.e., `onempobs0` is `1-pobs0`; (iii) argument `zero` has a new default so that the `onempobs0` is intercept-only by default. Now `zapoissonff()` is generally recommended over `zapoisson()`. Both functions implement Fisher scoring and can handle multiple responses.

Value

An object of class `"vglmff"` (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

The `fitted.values` slot of the fitted object, which should be extracted by the generic function `fitted`, returns the mean μ (default) which is given by

$$\mu = (1 - p_0)\lambda/[1 - \exp(-\lambda)].$$

If `type.fitted = "pobs0"` then p_0 is returned.

Note

There are subtle differences between this family function and zipoisson and yip88. In particular, zipoisson is a *mixture* model whereas `zapoisson()` and yip88 are *conditional* models.

Note this family function allows p_0 to be modelled as functions of the covariates.

This family function effectively combines pospoisson and binomialff into one family function. This family function can handle multiple responses, e.g., more than one species.

It is recommended that Gaitdpois be used, e.g., `rgaitdpois(nn, lambda, pobs.mlm = pobs0, a.mlm = 0)` instead of `rzapois(nn, lambda, pobs0 = pobs0)`.

Author(s)

T. W. Yee

References

Welsh, A. H., Cunningham, R. B., Donnelly, C. F. and Lindenmayer, D. B. (1996). Modelling the abundances of rare species: statistical models for counts with extra zeros. *Ecological Modelling*, **88**, 297–308.

Angers, J-F. and Biswas, A. (2003). A Bayesian analysis of zero-inflated generalized Poisson model. *Computational Statistics & Data Analysis*, **42**, 37–46.

Yee, T. W. (2014). Reduced-rank vector generalized linear models with two linear predictors. *Computational Statistics and Data Analysis*, **71**, 889–902.

See Also

Gaitdpois, rzapois, zipoisson, gaitdpoisson, pospoisson, posnegbinomial, spikeplot, binomialff, CommonVGAMffArguments, simulate.vlm.

Examples

```
zdata <- data.frame(x2 = runif(nn <- 1000))
zdata <- transform(zdata, pobs0  = logitlink( -1 + 1*x2, inverse = TRUE),
                          lambda = loglink(-0.5 + 2*x2, inverse = TRUE))
zdata <- transform(zdata, y = rgaitdpois(nn, lambda, pobs.mlm = pobs0,
                                        a.mlm = 0))

with(zdata, table(y))
fit <- vglm(y ~ x2, zapoisson, data = zdata, trace = TRUE)
fit <- vglm(y ~ x2, zapoisson, data = zdata, trace = TRUE, crit = "coef")
head(fitted(fit))
head(predict(fit))
head(predict(fit, untransform = TRUE))
coef(fit, matrix = TRUE)
summary(fit)

# Another example ------------------------------
# Data from Angers and Biswas (2003)
abdata <- data.frame(y = 0:7, w = c(182, 41, 12, 2, 2, 0, 0, 1))
abdata <- subset(abdata, w > 0)
Abdata <- data.frame(yy = with(abdata, rep(y, w)))
fit3 <- vglm(yy ~ 1, zapoisson, data = Abdata, trace = TRUE, crit = "coef")
coef(fit3, matrix = TRUE)
Coef(fit3)  # Estimate lambda (they get 0.6997 with SE 0.1520)
head(fitted(fit3), 1)
with(Abdata, mean(yy))  # Compare this with fitted(fit3)
```

---

zero — *The zero Argument in VGAM Family Functions*

Description

The `zero` argument allows users to conveniently model certain linear/additive predictors as intercept-only.

Details

Often a certain parameter needs to be modelled simply while other parameters in the model may be more complex, for example, the λ parameter in LMS-Box-Cox quantile regression should be modelled more simply compared to its μ parameter. Another example is the ξ parameter in a GEV distribution which is should be modelled simpler than its μ parameter. Using the `zero` argument allows this to be fitted conveniently without having to input all the constraint matrices explicitly.

The `zero` argument can be assigned an integer vector from the set `{1:M}` where `M` is the number of linear/additive predictors. Full details about constraint matrices can be found in the references. See `CommonVGAMffArguments` for more information.

Value

Nothing is returned. It is simply a convenient argument for constraining certain linear/additive predictors to be an intercept only.

Warning

The use of other arguments may conflict with the `zero` argument. For example, using `constraints` to input constraint matrices may conflict with the `zero` argument. Another example is the argument `parallel`. In general users should not assume any particular order of precedence when there is potential conflict of definition. Currently no checking for consistency is made.

The argument `zero` may be renamed in the future to something better.

Side Effects

The argument creates the appropriate constraint matrices internally.

Note

In all **VGAM** family functions `zero = NULL` means none of the linear/additive predictors are modelled as intercepts-only. Almost all **VGAM** family function have `zero = NULL` as the default, but there are some exceptions, e.g., `binom2.or`.

Typing something like `coef(fit, matrix = TRUE)` is a useful way to ensure that the `zero` argument has worked as expected.

Author(s)

T. W. Yee

References

Yee, T. W. and Wild, C. J. (1996). Vector generalized additive models. *Journal of the Royal Statistical Society, Series B, Methodological*, **58**, 481–493.

Yee, T. W. and Hastie, T. J. (2003). Reduced-rank vector generalized linear models. *Statistical Modelling*, **3**, 15–41.

See Also

`CommonVGAMffArguments`, `constraints`.

Examples

```
args(multinomial)
args(binom2.or)
args(gpd)

#LMS quantile regression example
fit <- vglm(BMI ~ sm.bs(age, df = 4), lms.bcg(zero = c(1, 3)),
            data = bmi.nz, trace = TRUE)
coef(fit, matrix = TRUE)
```

Zeta *The Zeta Distribution*

Description

Density, distribution function, quantile function and random generation for the zeta distribution.

Usage

```
dzeta(x, shape, log = FALSE)
pzeta(q, shape, lower.tail = TRUE)
qzeta(p, shape)
rzeta(n, shape)
```

Arguments

`x, q, p, n`	Same as `Poisson`.
`shape`	The positive shape parameter s.
`lower.tail, log`	Same meaning as in `Normal`.

Details

The density function of the zeta distribution is given by

$$y^{-s-1}/\zeta(s+1)$$

where $s > 0$, $y = 1, 2, \ldots$, and ζ is Riemann's zeta function.

Value

`dzeta` gives the density, `pzeta` gives the distribution function, `qzeta` gives the quantile function, and `rzeta` generates random deviates.

Note

`qzeta()` runs slower and slower as `shape` approaches 0 and `shape` approaches 1. The **VGAM** family function `zetaff` estimates the shape parameter s.

Author(s)

T. W. Yee

References

Johnson N. L., Kotz S., and Balakrishnan N. (1993). *Univariate Discrete Distributions*, 2nd ed. New York: Wiley.

See Also

zeta, zetaff, Oazeta, Oizeta, Otzeta.

Examples

```
dzeta(1:20, shape = 2)
myshape <- 0.5
max(abs(pzeta(1:200, myshape) -
    cumsum(1/(1:200)^(1+myshape)) / zeta(myshape+1)))  # Should be 0

## Not run:  plot(1:6, dzeta(1:6, 2), type = "h", las = 1,
               col = "orange", ylab = "Probability",
 main = "zeta probability function; orange: shape = 2; blue: shape = 1")
points(0.10 + 1:6, dzeta(1:6, 1), type = "h", col = "blue")
## End(Not run)
```

---

zeta *Riemann's (and the Hurwitz) Zeta Function, With Derivatives*

Description

Computes Riemann's zeta function and its first two derivatives. Also can compute the Hurwitz zeta function.

Usage

```
zeta(x, deriv = 0, shift = 1)
```

Arguments

x	A complex-valued vector/matrix whose real values must be ≥ 1. Otherwise, x may be real. It is called s below. If deriv is 1 or 2 then x must be real and positive.
deriv	An integer equalling 0 or 1 or 2, which is the order of the derivative. The default means it is computed ordinarily.
shift	Positive and numeric, called A below. Allows for the Hurwitz zeta to be returned. The default corresponds to the Riemann formula.

Details

The (Riemann) formula for real s is

$$\sum_{n=1}^{\infty} 1/n^s.$$

While the usual definition involves an infinite series that converges when the real part of the argument is > 1, more efficient methods have been devised to compute the value. In particular, this function uses Euler-Maclaurin summation. Theoretically, the zeta function can be computed over the whole complex plane because of analytic continuation.

The (Riemann) formula used here for analytic continuation is

$$\zeta(s) = 2^s \pi^{s-1} \sin(\pi s/2)\Gamma(1-s)\zeta(1-s).$$

This is actually one of several formulas, but this one was discovered by Riemann himself and is called the *functional equation*.

The Hurwitz zeta function for real $s > 0$ is

$$\sum_{n=0}^{\infty} 1/(A+n)^s.$$

where $0 < A$ is known here as the `shift`. Since $A = 1$ by default, this function will therefore return Riemann's zeta function by default. Currently derivatives are unavailable.

Value

The default is a vector/matrix of computed values of Riemann's zeta function. If `shift` contains values not equal to 1, then this is Hurwitz's zeta function.

Warning

This function has not been fully tested, especially the derivatives. In particular, analytic continuation does not work here for complex `x` with `Re(x)<1` because currently the `gamma` function does not handle complex arguments.

Note

Estimation of the parameter of the zeta distribution can be achieved with `zetaff`.

Author(s)

T. W. Yee, with the help of Garry J. Tee.

References

Riemann, B. (1859). Ueber die Anzahl der Primzahlen unter einer gegebenen Grosse. *Monatsberichte der Berliner Akademie, November 1859*.

Edwards, H. M. (1974). *Riemann's Zeta Function*. Academic Press: New York.

Markman, B. (1965). The Riemann zeta function. *BIT*, **5**, 138–141.

Abramowitz, M. and Stegun, I. A. (1972). *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, New York: Dover Publications Inc.

See Also

zetaff, Zeta, oazeta, oizeta, otzeta, lerch, gamma.

Examples

```
zeta(2:10)

## Not run:
curve(zeta, -13, 0.8, xlim = c(-12, 10), ylim = c(-1, 4), col = "orange",
      las = 1, main = expression({zeta}(x)))
curve(zeta, 1.2,  12, add = TRUE, col = "orange")
abline(v = 0, h = c(0, 1), lty = "dashed", col = "gray")

curve(zeta, -14, -0.4, col = "orange", main = expression({zeta}(x)))
abline(v = 0, h = 0, lty = "dashed", col = "gray")  # Close up plot

x <- seq(0.04, 0.8, len = 100)  # Plot of the first derivative
plot(x, zeta(x, deriv = 1), type = "l", las = 1, col = "blue",
     xlim = c(0.04, 3), ylim = c(-6, 0), main = "zeta'(x)")
x <- seq(1.2, 3, len = 100)
lines(x, zeta(x, deriv = 1), col = "blue")
abline(v = 0, h = 0, lty = "dashed", col = "gray")
## End(Not run)

zeta(2) - pi^2 / 6     # Should be 0
zeta(4) - pi^4 / 90    # Should be 0
zeta(6) - pi^6 / 945   # Should be 0
zeta(8) - pi^8 / 9450  # Should be 0
zeta(0, deriv = 1) + 0.5 * log(2*pi)  # Should be 0
gamma0 <-  0.5772156649
gamma1 <- -0.07281584548
zeta(0, deriv = 2) -
  gamma1 + 0.5 * (log(2*pi))^2 + pi^2/24 - gamma0^2 / 2  # Should be 0
zeta(0.5, deriv = 1) + 3.92264613  # Should be 0
zeta(2.0, deriv = 1) + 0.93754825431  # Should be 0
```

---

zetaff *Zeta Distribution Family Function*

Description

Estimates the parameter of the zeta distribution.

Usage

```
zetaff(lshape = "loglink", ishape = NULL, gshape = 1 + exp(-seq(7)),
       zero = NULL)
```

Arguments

`lshape, ishape, zero`
: These arguments apply to the (positive) parameter p. See `Links` for more choices. Choosing `loglog` constrains $p > 1$, but may fail if the maximum likelihood estimate is less than one. See `CommonVGAMffArguments` for more information.

`gshape`
: See `CommonVGAMffArguments` for more information.

Details

In this long tailed distribution the response must be a positive integer. The probability function for a response Y is

$$P(Y = y) = 1/[y^{p+1}\zeta(p + 1)], \quad p > 0, \quad y = 1, 2, ...$$

where ζ is Riemann's zeta function. The parameter p is positive, therefore a log link is the default. The mean of Y is $\mu = \zeta(p)/\zeta(p+1)$ (provided $p > 1$) and these are the fitted values. The variance of Y is $\zeta(p - 1)/\zeta(p + 1) - \mu^2$ provided $p > 2$.

It appears that good initial values are needed for successful convergence. If convergence is not obtained, try several values ranging from values near 0 to values about 10 or more.

Multiple responses are handled.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Note

The `zeta` function may be used to compute values of the zeta function.

Author(s)

T. W. Yee

References

pp.527– of Chapter 11 of Johnson N. L., Kemp, A. W. and Kotz S. (2005). *Univariate Discrete Distributions*, 3rd edition, Hoboken, New Jersey: Wiley.

Knight, K. (2000). *Mathematical Statistics*. Boca Raton, FL, USA: Chapman & Hall/CRC Press.

See Also

`zeta`, `Zeta`, `gaitdzeta`, `oazeta`, `oizeta`, `otzeta`, `diffzeta`, `hzeta`, `zipf`.

Examples

```
zdata <- data.frame(y = 1:5, w =  c(63, 14, 5, 1, 2))  # Knight, p.304
fit <- vglm(y ~ 1, zetaff, data = zdata, trace = TRUE, weight = w, crit = "c")
(phat <- Coef(fit))  # 1.682557
with(zdata, cbind(round(dzeta(y, phat) * sum(w), 1), w))

with(zdata, weighted.mean(y, w))
fitted(fit, matrix = FALSE)
predict(fit)

# The following should be zero at the MLE:
with(zdata, mean(log(rep(y, w))) + zeta(1+phat, deriv = 1) / zeta(1+phat))
```

Zibinom *Zero-Inflated Binomial Distribution*

Description

Density, distribution function, quantile function and random generation for the zero-inflated binomial distribution with parameter `pstr0`.

Usage

```
dzibinom(x, size, prob, pstr0 = 0, log = FALSE)
pzibinom(q, size, prob, pstr0 = 0)
qzibinom(p, size, prob, pstr0 = 0)
rzibinom(n, size, prob, pstr0 = 0)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`size`	number of trials. It is the N symbol in the formula given in `zibinomial`.
`prob`	probability of success on each trial.
`n`	Same as in `runif`.
`log`	Same as `pbinom`.
`pstr0`	Probability of a structural zero (i.e., ignoring the binomial distribution), called ϕ. The default value of $\phi = 0$ corresponds to the response having an ordinary binomial distribution.

Details

The probability function of Y is 0 with probability ϕ, and $Binomial(size, prob)$ with probability $1 - \phi$. Thus

$$P(Y = 0) = \phi + (1 - \phi)P(W = 0)$$

where W is distributed $Binomial(size, prob)$.

Value

`dzibinom` gives the density, `pzibinom` gives the distribution function, `qzibinom` gives the quantile function, and `rzibinom` generates random deviates.

Note

The argument `pstr0` is recycled to the required length, and must have values which lie in the interval $[0, 1]$.

These functions actually allow for *zero-deflation*. That is, the resulting probability of a zero count is *less than* the nominal value of the parent distribution. See `Zipois` for more information.

Author(s)

T. W. Yee

See Also

`zibinomial`, `Gaitdbinom`, `Binomial`.

Examples

```
prob <- 0.2; size <- 10; pstr0 <- 0.5
(ii <- dzibinom(0:size, size, prob, pstr0 = pstr0))
max(abs(cumsum(ii) - pzibinom(0:size, size, prob, pstr0 = pstr0)))  # 0?
table(rzibinom(100, size, prob, pstr0 = pstr0))

table(qzibinom(runif(100), size, prob, pstr0 = pstr0))
round(dzibinom(0:10, size, prob, pstr0 = pstr0) * 100)  # Similar?

## Not run:  x <- 0:size
barplot(rbind(dzibinom(x, size, prob, pstr0 = pstr0),
                dbinom(x, size, prob)),
    beside = TRUE, col = c("blue", "green"), ylab = "Probability",
    main = paste0("ZIB(", size, ", ", prob, ", pstr0 = ", pstr0, ")",
                 " (blue) vs Binomial(", size, ", ", prob, ") (green)"),
    names.arg = as.character(x), las = 1, lwd = 2)
## End(Not run)
```

`zibinomial` *Zero-Inflated Binomial Distribution Family Function*

Description

Fits a zero-inflated binomial distribution by maximum likelihood estimation.

Usage

```
zibinomial(lpstr0 = "logitlink", lprob = "logitlink",
           type.fitted = c("mean", "prob", "pobs0", "pstr0", "onempstr0"),
           ipstr0 = NULL, zero = NULL, multiple.responses = FALSE,
           imethod = 1)
zibinomialff(lprob = "logitlink", lonempstr0 = "logitlink",
             type.fitted = c("mean", "prob", "pobs0", "pstr0", "onempstr0"),
             ionempstr0 = NULL, zero = "onempstr0",
             multiple.responses = FALSE, imethod = 1)
```

Arguments

- `lpstr0, lprob` — Link functions for the parameter ϕ and the usual binomial probability μ parameter. See `Links` for more choices. For the zero-*deflated* model see below.
- `type.fitted` — See `CommonVGAMffArguments` and `fittedvlm`.
- `ipstr0` — Optional initial values for ϕ, whose values must lie between 0 and 1. The default is to compute an initial value internally. If a vector then recyling is used.
- `lonempstr0, ionempstr0` — Corresponding arguments for the other parameterization. See details below.
- `multiple.responses` — Logical. Currently it must be `FALSE` to mean the function does not handle multiple responses. This is to remain compatible with the same argument in `binomialff`.
- `zero, imethod` — See `CommonVGAMffArguments` for information. Argument `zero` changed its default value for version 0.9-2.

Details

These functions are based on

$$P(Y=0) = \phi + (1-\phi)(1-\mu)^N,$$

for $y=0$, and

$$P(Y=y) = (1-\phi)\binom{N}{Ny}\mu^{Ny}(1-\mu)^{N(1-y)}.$$

for $y = 1/N, 2/N, \ldots, 1$. That is, the response is a sample proportion out of N trials, and the argument `size` in `rzibinom` is N here. The parameter ϕ is the probability of a structural zero, and it satisfies $0 < \phi < 1$. The mean of Y is $E(Y) = (1-\phi)\mu$ and these are returned as the fitted values by default. By default, the two linear/additive predictors for `zibinomial()` are $(logit(\phi), logit(\mu))^T$.

The **VGAM** family function `zibinomialff()` has a few changes compared to `zibinomial()`. These are: (i) the order of the linear/additive predictors is switched so the binomial probability comes first; (ii) argument `onempstr0` is now 1 minus the probability of a structural zero, i.e., the probability of the parent (binomial) component, i.e., `onempstr0` is `1-pstr0`; (iii) argument `zero` has a new default so that the `onempstr0` is intercept-only by default. Now `zibinomialff()` is generally recommended over `zibinomial()`. Both functions implement Fisher scoring.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm and vgam.

Warning

Numerical problems can occur. Half-stepping is not uncommon. If failure to converge occurs, make use of the argument ipstr0 or ionempstr0, or imethod.

Note

The response variable must have one of the formats described by binomialff, e.g., a factor or two column matrix or a vector of sample proportions with the weights argument specifying the values of N.

To work well, one needs large values of N and $\mu > 0$, i.e., the larger N and μ are, the better. If $N = 1$ then the model is unidentifiable since the number of parameters is excessive.

Setting stepsize = 0.5, say, may aid convergence.

Estimated probabilities of a structural zero and an observed zero are returned, as in zipoisson.

The zero-*deflated* binomial distribution might be fitted by setting lpstr0 = identitylink, albeit, not entirely reliably. See zipoisson for information that can be applied here. Else try the zero-altered binomial distribution (see zabinomial).

Author(s)

T. W. Yee

References

Welsh, A. H., Lindenmayer, D. B. and Donnelly, C. F. (2013). Fitting and interpreting occupancy models. *PLOS One*, **8**, 1–21.

See Also

rzibinom, binomialff, posbinomial, spikeplot, Binomial.

Examples

```
size <- 10  # Number of trials; N in the notation above
nn <- 200
zdata <- data.frame(pstr0 = logitlink( 0, inverse = TRUE),  # 0.50
                    mubin = logitlink(-1, inverse = TRUE),  # Mean of usual binomial
                    sv    = rep(size, length = nn))
zdata <- transform(zdata,
                   y = rzibinom(nn, size = sv, prob = mubin, pstr0 = pstr0))
with(zdata, table(y))
fit <- vglm(cbind(y, sv - y) ~ 1, zibinomialff, data = zdata, trace = TRUE)
fit <- vglm(cbind(y, sv - y) ~ 1, zibinomialff, data = zdata, trace = TRUE,
            stepsize = 0.5)
```

```
coef(fit, matrix = TRUE)
Coef(fit)  # Useful for intercept-only models
head(fitted(fit, type = "pobs0"))  # Estimate of P(Y = 0)
head(fitted(fit))
with(zdata, mean(y))  # Compare this with fitted(fit)
summary(fit)
```

Zigeom *Zero-Inflated Geometric Distribution*

Description

Density, and random generation for the zero-inflated geometric distribution with parameter `pstr0`.

Usage

```
dzigeom(x, prob, pstr0 = 0, log = FALSE)
pzigeom(q, prob, pstr0 = 0)
qzigeom(p, prob, pstr0 = 0)
rzigeom(n, prob, pstr0 = 0)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`prob`	see `dgeom`.
`n`	Same as in `runif`.
`pstr0`	Probability of structural zero (ignoring the geometric distribution), called ϕ. The default value corresponds to the response having an ordinary geometric distribution.
`log`	Logical. Return the logarithm of the answer?

Details

The probability function of Y is 0 with probability ϕ, and $geometric(prob)$ with probability $1-\phi$. Thus

$$P(Y = 0) = \phi + (1-\phi)P(W = 0)$$

where W is distributed $geometric(prob)$.

Value

`dzigeom` gives the density, `pzigeom` gives the distribution function, `qzigeom` gives the quantile function, and `rzigeom` generates random deviates.

Note

The argument pstr0 is recycled to the required length, and must have values which lie in the interval $[0, 1]$.

These functions actually allow for *zero-deflation*. That is, the resulting probability of a zero count is *less than* the nominal value of the parent distribution. See Zipois for more information.

Author(s)

T. W. Yee

See Also

zigeometric, dgeom.

Examples

```
prob <- 0.5; pstr0 <- 0.2; x <- (-1):20
(ii <- dzigeom(x, prob, pstr0))
max(abs(cumsum(ii) - pzigeom(x, prob, pstr0)))  # Should be 0
table(rzigeom(1000, prob, pstr0))

## Not run:  x <- 0:10
barplot(rbind(dzigeom(x, prob, pstr0), dgeom(x, prob)),
   beside = TRUE, col = c("blue","orange"),
   ylab = "P[Y = y]", xlab = "y", las = 1,
   main = paste0("zigeometric(", prob, ", pstr0 = ", pstr0,
                 ") (blue) vs", " geometric(", prob, ") (orange)"),
   names.arg = as.character(x))
## End(Not run)
```

---

zigeometric *Zero-Inflated Geometric Distribution Family Function*

Description

Fits a zero-inflated geometric distribution by maximum likelihood estimation.

Usage

```
zigeometric(lpstr0  = "logitlink", lprob = "logitlink",
            type.fitted = c("mean", "prob", "pobs0", "pstr0", "onempstr0"),
            ipstr0  = NULL, iprob = NULL,
            imethod = 1, bias.red = 0.5, zero = NULL)
zigeometricff(lprob = "logitlink", lonempstr0 = "logitlink",
              type.fitted = c("mean", "prob", "pobs0", "pstr0", "onempstr0"),
              iprob = NULL, ionempstr0 = NULL,
              imethod = 1, bias.red = 0.5, zero = "onempstr0")
```

Arguments

`lpstr0, lprob` Link functions for the parameters ϕ and p (`prob`). The usual geometric probability parameter is the latter. The probability of a structural zero is the former. See `Links` for more choices. For the zero-*deflated* model see below.

`lonempstr0, ionempstr0` Corresponding arguments for the other parameterization. See details below.

`bias.red` A constant used in the initialization process of `pstr0`. It should lie between 0 and 1, with 1 having no effect.

`type.fitted` See `CommonVGAMffArguments` and `fittedvlm` for information.

`ipstr0, iprob` See `CommonVGAMffArguments` for information.

`zero, imethod` See `CommonVGAMffArguments` for information.

Details

Function `zigeometric()` is based on

$$P(Y = 0) = \phi + (1 - \phi)p,$$

for $y = 0$, and

$$P(Y = y) = (1 - \phi)p(1 - p)^y.$$

for $y = 1, 2, \ldots$. The parameter ϕ satisfies $0 < \phi < 1$. The mean of Y is $E(Y) = (1-\phi)p/(1-p)$ and these are returned as the fitted values by default. By default, the two linear/additive predictors are $(logit(\phi), logit(p))^T$. Multiple responses are handled.

Estimated probabilities of a structural zero and an observed zero can be returned, as in `zipoisson`; see `fittedvlm` for information.

The **VGAM** family function `zigeometricff()` has a few changes compared to `zigeometric()`. These are: (i) the order of the linear/additive predictors is switched so the geometric probability comes first; (ii) argument `onempstr0` is now 1 minus the probability of a structural zero, i.e., the probability of the parent (geometric) component, i.e., `onempstr0` is `1-pstr0`; (iii) argument `zero` has a new default so that the `onempstr0` is intercept-only by default. Now `zigeometricff()` is generally recommended over `zigeometric()`. Both functions implement Fisher scoring and can handle multiple responses.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Note

The zero-*deflated* geometric distribution might be fitted by setting `lpstr0 = identitylink`, albeit, not entirely reliably. See `zipoisson` for information that can be applied here. Else try the zero-altered geometric distribution (see `zageometric`).

Author(s)

T. W. Yee

See Also

rzigeom, geometric, zageometric, spikeplot, rgeom, simulate.vlm.

Examples

```
gdata <- data.frame(x2 = runif(nn <- 1000) - 0.5)
gdata <- transform(gdata, x3 = runif(nn) - 0.5,
                          x4 = runif(nn) - 0.5)
gdata <- transform(gdata, eta1 =  1.0 - 1.0 * x2 + 2.0 * x3,
                          eta2 = -1.0,
                          eta3 =  0.5)
gdata <- transform(gdata, prob1 = logitlink(eta1, inverse = TRUE),
                          prob2 = logitlink(eta2, inverse = TRUE),
                          prob3 = logitlink(eta3, inverse = TRUE))
gdata <- transform(gdata, y1 = rzigeom(nn, prob1, pstr0 = prob3),
                          y2 = rzigeom(nn, prob2, pstr0 = prob3),
                          y3 = rzigeom(nn, prob2, pstr0 = prob3))
with(gdata, table(y1))
with(gdata, table(y2))
with(gdata, table(y3))
head(gdata)

fit1 <- vglm(y1 ~ x2 + x3 + x4, zigeometric(zero = 1), data = gdata, trace = TRUE)
coef(fit1, matrix = TRUE)
head(fitted(fit1, type = "pstr0"))

fit2 <- vglm(cbind(y2, y3) ~ 1, zigeometric(zero = 1), data = gdata, trace = TRUE)
coef(fit2, matrix = TRUE)
summary(fit2)
```

Zinegbin *Zero-Inflated Negative Binomial Distribution*

Description

Density, distribution function, quantile function and random generation for the zero-inflated negative binomial distribution with parameter pstr0.

Usage

```
dzinegbin(x, size, prob = NULL, munb = NULL, pstr0 = 0, log = FALSE)
pzinegbin(q, size, prob = NULL, munb = NULL, pstr0 = 0)
qzinegbin(p, size, prob = NULL, munb = NULL, pstr0 = 0)
rzinegbin(n, size, prob = NULL, munb = NULL, pstr0 = 0)
```

Arguments

`x, q` vector of quantiles.

`p` vector of probabilities.

`n` Same as in `runif`.

`size, prob, munb, log`
Arguments matching `dnbinom`. The argument `munb` corresponds to `mu` in `dnbinom` and has been renamed to emphasize the fact that it is the mean of the negative binomial *component*.

`pstr0` Probability of structural zero (i.e., ignoring the negative binomial distribution), called ϕ.

Details

The probability function of Y is 0 with probability ϕ, and a negative binomial distribution with probability $1 - \phi$. Thus

$$P(Y = 0) = \phi + (1 - \phi)P(W = 0)$$

where W is distributed as a negative binomial distribution (see `rnbinom`.) See `negbinomial`, a **VGAM** family function, for the formula of the probability density function and other details of the negative binomial distribution.

Value

`dzinegbin` gives the density, `pzinegbin` gives the distribution function, `qzinegbin` gives the quantile function, and `rzinegbin` generates random deviates.

Note

The argument `pstr0` is recycled to the required length, and must have values which lie in the interval $[0, 1]$.

These functions actually allow for *zero-deflation*. That is, the resulting probability of a zero count is *less than* the nominal value of the parent distribution. See `Zipois` for more information.

Author(s)

T. W. Yee

See Also

`zinegbinomial`, `rnbinom`, `rzipois`.

Examples

```
munb <- 3; pstr0 <- 0.2; size <- k <- 10; x <- 0:10
(ii <- dzinegbin(x, pstr0 = pstr0, mu = munb, size = k))
max(abs(cumsum(ii) - pzinegbin(x, pstr0 = pstr0, mu = munb, size = k)))
table(rzinegbin(100, pstr0 = pstr0, mu = munb, size = k))

table(qzinegbin(runif(1000), pstr0 = pstr0, mu = munb, size = k))
```

```
round(dzinegbin(x, pstr0 = pstr0, mu = munb, size = k) * 1000)  # Similar?

## Not run: barplot(rbind(dzinegbin(x, pstr0 = pstr0, mu = munb, size = k),
                dnbinom(x, mu = munb, size = k)), las = 1,
    beside = TRUE, col = c("blue", "green"), ylab = "Probability",
    main = paste("ZINB(mu = ", munb, ", k = ", k, ", pstr0 = ", pstr0,
                 ") (blue) vs NB(mu = ", munb,
                 ", size = ", k, ") (green)", sep = ""),
    names.arg = as.character(x))
## End(Not run)
```

`zinegbinomial` *Zero-Inflated Negative Binomial Distribution Family Function*

Description

Fits a zero-inflated negative binomial distribution by full maximum likelihood estimation.

Usage

```
zinegbinomial(zero = "size",
               type.fitted = c("mean", "munb", "pobs0", "pstr0",
               "onempstr0"),
               mds.min = 1e-3, nsimEIM = 500, cutoff.prob = 0.999,
               eps.trig = 1e-7, max.support = 4000, max.chunk.MB = 30,
               lpstr0 = "logitlink", lmunb = "loglink", lsize = "loglink",
               imethod = 1, ipstr0 = NULL, imunb =  NULL,
               iprobs.y = NULL, isize = NULL,
               gprobs.y = (0:9)/10,
               gsize.mux = exp(c(-30, -20, -15, -10, -6:3)))
zinegbinomialff(lmunb = "loglink", lsize = "loglink", lonempstr0 = "logitlink",
                 type.fitted = c("mean", "munb", "pobs0", "pstr0",
                 "onempstr0"), imunb = NULL, isize = NULL, ionempstr0 =
                 NULL, zero = c("size", "onempstr0"), imethod = 1,
                 iprobs.y = NULL, cutoff.prob = 0.999,
                 eps.trig = 1e-7,  max.support = 4000, max.chunk.MB = 30,
                 gprobs.y = (0:9)/10, gsize.mux = exp((-12:6)/2),
                 mds.min = 1e-3, nsimEIM = 500)
```

Arguments

`lpstr0, lmunb, lsize`
: Link functions for the parameters ϕ, the mean and k; see `negbinomial` for details, and `Links` for more choices. For the zero-*deflated* model see below.

`type.fitted`
: See `CommonVGAMffArguments` and `fittedvlm` for more information.

`ipstr0, isize, imunb`
: Optional initial values for ϕ and k and μ. The default is to compute an initial value internally for both. If a vector then recycling is used.

lonempstr0, ionempstr0
: Corresponding arguments for the other parameterization. See details below.

imethod
: An integer with value 1 or 2 or 3 which specifies the initialization method for the mean parameter. If failure to converge occurs try another value. See CommonVGAMffArguments for more information.

zero
: Specifies which linear/additive predictors are to be modelled as intercept-only. They can be such that their absolute values are either 1 or 2 or 3. The default is the ϕ and k parameters (both for each response). See CommonVGAMffArguments for more information.

nsimEIM
: See CommonVGAMffArguments for information.

iprobs.y, cutoff.prob, max.support, max.chunk.MB
: See negbinomial and/or posnegbinomial for details.

mds.min, eps.trig
: See negbinomial for details.

gprobs.y, gsize.mux
: These arguments relate to grid searching in the initialization process. See negbinomial and/or posnegbinomial for details.

Details

These functions are based on

$$P(Y = 0) = \phi + (1 - \phi)(k/(k + \mu))^k,$$

and for $y = 1, 2, \ldots,$

$$P(Y = y) = (1 - \phi)\, dnbinom(y, \mu, k).$$

The parameter ϕ satisfies $0 < \phi < 1$. The mean of Y is $(1 - \phi)\mu$ (returned as the fitted values). By default, the three linear/additive predictors for `zinegbinomial()` are $(logit(\phi), \log(\mu), \log(k))^T$. See negbinomial, another **VGAM** family function, for the formula of the probability density function and other details of the negative binomial distribution.

Independent multiple responses are handled. If so then arguments `ipstr0` and `isize` may be vectors with length equal to the number of responses.

The **VGAM** family function `zinegbinomialff()` has a few changes compared to `zinegbinomial()`. These are: (i) the order of the linear/additive predictors is switched so the NB mean comes first; (ii) `onempstr0` is now 1 minus the probability of a structural 0, i.e., the probability of the parent (NB) component, i.e., `onempstr0` is `1-pstr0`; (iii) argument `zero` has a new default so that the `onempstr0` is intercept-only by default. Now `zinegbinomialff()` is generally recommended over `zinegbinomial()`. Both functions implement Fisher scoring and can handle multiple responses.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm, and vgam.

Warning

This model can be difficult to fit to data, and this family function is fragile. The model is especially difficult to fit reliably when the estimated k parameter is very large (so the model approaches a zero-inflated Poisson distribution) or much less than 1 (and gets more difficult as it approaches 0). Numerical problems can also occur, e.g., when the probability of a zero is actually less than, and not more than, the nominal probability of zero. Similarly, numerical problems can occur if there is little or no 0-inflation, or when the sample size is small. Half-stepping is not uncommon. Successful convergence is sensitive to the initial values, therefore if failure to converge occurs, try using combinations of arguments `stepsize` (in `vglm.control`), `imethod`, `imunb`, `ipstr0`, `isize`, and/or `zero` if there are explanatory variables. Else try fitting an ordinary `negbinomial` model or a `zipoisson` model.

This **VGAM** family function can be computationally expensive and can run slowly; setting `trace = TRUE` is useful for monitoring convergence.

Note

Estimated probabilities of a structural zero and an observed zero can be returned, as in `zipoisson`; see `fittedvlm` for more information.

If k is large then the use of **VGAM** family function `zipoisson` is probably preferable. This follows because the Poisson is the limiting distribution of a negative binomial as k tends to infinity.

The zero-*deflated* negative binomial distribution might be fitted by setting `lpstr0 = identitylink`, albeit, not entirely reliably. See `zipoisson` for information that can be applied here. Else try the zero-altered negative binomial distribution (see `zanegbinomial`).

Author(s)

T. W. Yee

See Also

`gaitdnbinomial`, `Zinegbin`, `negbinomial`, `spikeplot`, `rpois`, `CommonVGAMffArguments`.

Examples

```
## Not run:
# Example 1
ndata <- data.frame(x2 = runif(nn <- 1000))
ndata <- transform(ndata, pstr0 = logitlink(-0.5 + 1 * x2, inverse = TRUE),
                          munb  =   exp( 3   + 1 * x2),
                          size  =   exp( 0   + 2 * x2))
ndata <- transform(ndata,
                   y1 = rzinegbin(nn, mu = munb, size = size, pstr0 = pstr0))
with(ndata, table(y1)["0"] / sum(table(y1)))
nfit <- vglm(y1 ~ x2, zinegbinomial(zero = NULL), data = ndata)
coef(nfit, matrix = TRUE)
summary(nfit)
head(cbind(fitted(nfit), with(ndata, (1 - pstr0) * munb)))
round(vcov(nfit), 3)
```

```
# Example 2: RR-ZINB could also be called a COZIVGLM-ZINB-2
ndata <- data.frame(x2 = runif(nn <- 2000))
ndata <- transform(ndata, x3 = runif(nn))
ndata <- transform(ndata, eta1 =          3   + 1   * x2 + 2 * x3)
ndata <- transform(ndata, pstr0  = logitlink(-1.5 + 0.5 * eta1, inverse = TRUE),
                          munb = exp(eta1),
                          size = exp(4))
ndata <- transform(ndata,
                   y1 = rzinegbin(nn, pstr0 = pstr0, mu = munb, size = size))
with(ndata, table(y1)["0"] / sum(table(y1)))
rrzinb <- rrvglm(y1 ~ x2 + x3, zinegbinomial(zero = NULL), data = ndata,
                 Index.corner = 2, str0 = 3, trace = TRUE)
coef(rrzinb, matrix = TRUE)
Coef(rrzinb)

## End(Not run)
```

---

zipebcom *Exchangeable Bivariate cloglog Odds-ratio Model From a Zero-inflated Poisson Distribution*

Description

Fits an exchangeable bivariate odds-ratio model to two binary responses with a complementary log-log link. The data are assumed to come from a zero-inflated Poisson distribution that has been converted to presence/absence.

Usage

```
zipebcom(lmu12 = "clogloglink", lphi12 = "logitlink", loratio = "loglink",
         imu12 = NULL, iphi12 = NULL, ioratio = NULL,
         zero = c("phi12", "oratio"), tol = 0.001, addRidge = 0.001)
```

Arguments

`lmu12, imu12` Link function, extra argument and optional initial values for the first (and second) marginal probabilities. Argument `lmu12` should be left alone. Argument `imu12` may be of length 2 (one element for each response).

`lphi12` Link function applied to the ϕ parameter of the zero-inflated Poisson distribution (see `zipoisson`). See `Links` for more choices.

`loratio` Link function applied to the odds ratio. See `Links` for more choices.

`iphi12, ioratio` Optional initial values for ϕ and the odds ratio. See `CommonVGAMffArguments` for more details. In general, good initial values (especially for `iphi12`) are often required, therefore use these arguments if convergence failure occurs. If inputted, the value of `iphi12` cannot be more than the sample proportions of zeros in either response.

`zero` Which linear/additive predictor is modelled as an intercept only? A `NULL` means none. The default has both ϕ and the odds ratio as not being modelled as a function of the explanatory variables (apart from an intercept). See `CommonVGAMffArguments` for information.

`tol` Tolerance for testing independence. Should be some small positive numerical value.

`addRidge` Some small positive numerical value. The first two diagonal elements of the working weight matrices are multiplied by `1+addRidge` to make it diagonally dominant, therefore positive-definite.

Details

This **VGAM** family function fits an exchangeable bivariate odds ratio model (`binom2.or`) with a `clogloglink` link. The data are assumed to come from a zero-inflated Poisson (ZIP) distribution that has been converted to presence/absence. Explicitly, the default model is

$$cloglog[P(Y_j = 1)/(1 - \phi)] = \eta_1, \quad j = 1, 2$$

for the (exchangeable) marginals, and

$$logit[\phi] = \eta_2,$$

for the mixing parameter, and

$$\log[P(Y_{00} = 1)P(Y_{11} = 1)/(P(Y_{01} = 1)P(Y_{10} = 1))] = \eta_3,$$

specifies the dependency between the two responses. Here, the responses equal 1 for a success and a 0 for a failure, and the odds ratio is often written $\psi = p_{00}p_{11}/(p_{10}p_{01})$. We have $p_{10} = p_{01}$ because of the exchangeability.

The second linear/additive predictor models the ϕ parameter (see `zipoisson`). The third linear/additive predictor is the same as `binom2.or`, viz., the log odds ratio.

Suppose a dataset1 comes from a Poisson distribution that has been converted to presence/absence, and that both marginal probabilities are the same (exchangeable). Then `binom2.or("clogloglink", exch=TRUE)` is appropriate. Now suppose a dataset2 comes from a *zero-inflated* Poisson distribution. The first linear/additive predictor of `zipebcom()` applied to dataset2 is the same as that of `binom2.or("clogloglink", exch=TRUE)` applied to dataset1. That is, the ϕ has been taken care of by `zipebcom()` so that it is just like the simpler `binom2.or`.

Note that, for η_1, `mu12 = prob12 / (1-phi12)` where `prob12` is the probability of a 1 under the ZIP model. Here, `mu12` correspond to `mu1` and `mu2` in the `binom2.or`-Poisson model.

If $\phi = 0$ then `zipebcom()` should be equivalent to `binom2.or("clogloglink", exch=TRUE)`. Full details are given in Yee and Dirnbock (2009).

The leading 2×2 submatrix of the expected information matrix (EIM) is of rank-1, not 2! This is due to the fact that the parameters corresponding to the first two linear/additive predictors are unidentifiable. The quick fix around this problem is to use the `addRidge` adjustment. The model is fitted by maximum likelihood estimation since the full likelihood is specified. Fisher scoring is implemented.

The default models η_2 and η_3 as single parameters only, but this can be circumvented by setting `zero=NULL` in order to model the ϕ and odds ratio as a function of all the explanatory variables.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm and vgam.

When fitted, the fitted.values slot of the object contains the four joint probabilities, labelled as (Y_1, Y_2) = (0,0), (0,1), (1,0), (1,1), respectively. These estimated probabilities should be extracted with the fitted generic function.

Warning

The fact that the EIM is not of full rank may mean the model is naturally ill-conditioned. Not sure whether there are any negative consequences wrt theory. For now it is certainly safer to fit binom2.or to bivariate binary responses.

Note

The "12" in the argument names reinforce the user about the exchangeability assumption. The name of this **VGAM** family function stands for *zero-inflated Poisson exchangeable bivariate complementary log-log odds-ratio model* or ZIP-EBCOM.

See binom2.or for details that are pertinent to this **VGAM** family function too. Even better initial values are usually needed here.

The xij (see vglm.control) argument enables environmental variables with different values at the two time points to be entered into an exchangeable binom2.or model. See the author's webpage for sample code.

References

Yee, T. W. and Dirnbock, T. (2009). Models for analysing species' presence/absence data at two time points. Journal of Theoretical Biology, **259**(4), 684–694.

See Also

binom2.or, zipoisson, clogloglink, CommonVGAMffArguments.

Examples

```
zdata <- data.frame(x2 = seq(0, 1, len = (nsites <- 2000)))
zdata <- transform(zdata, eta1 =  -3 + 5 * x2,
                         phi1 = logitlink(-1, inverse = TRUE),
                         oratio = exp(2))
zdata <- transform(zdata, mu12 = clogloglink(eta1, inverse = TRUE) * (1-phi1))
tmat <-  with(zdata, rbinom2.or(nsites, mu1 = mu12, oratio = oratio, exch = TRUE))
zdata <- transform(zdata, ybin1 = tmat[, 1], ybin2 = tmat[, 2])

with(zdata, table(ybin1, ybin2)) / nsites  # For interest only
## Not run:
# Various plots of the data, for interest only
par(mfrow = c(2, 2))
plot(jitter(ybin1) ~ x2, data = zdata, col = "blue")
```

```
plot(jitter(ybin2) ~ jitter(ybin1), data = zdata, col = "blue")

plot(mu12 ~ x2, data = zdata, col = "blue", type = "l", ylim = 0:1,
     ylab = "Probability", main = "Marginal probability and phi")
with(zdata, abline(h = phi1[1], col = "red", lty = "dashed"))

tmat2 <- with(zdata, dbinom2.or(mu1 = mu12, oratio = oratio, exch = TRUE))
with(zdata, matplot(x2, tmat2, col = 1:4, type = "l", ylim = 0:1,
     ylab = "Probability", main = "Joint probabilities"))
## End(Not run)

# Now fit the model to the data.
fit <- vglm(cbind(ybin1, ybin2) ~ x2, zipebcom, data = zdata, trace = TRUE)
coef(fit, matrix = TRUE)
summary(fit)
vcov(fit)
```

Zipf *The Zipf Distribution*

Description

Density, distribution function, quantile function and random generation for the Zipf distribution.

Usage

```
dzipf(x, N, shape, log = FALSE)
pzipf(q, N, shape, log.p = FALSE)
qzipf(p, N, shape)
rzipf(n, N, shape)
```

Arguments

`x, q, p, n`	Same as `Poisson`.
`N, shape`	the number of elements, and the exponent characterizing the distribution. See `zipf` for more details.
`log, log.p`	Same meaning as in `Normal`.

Details

This is a finite version of the zeta distribution. See `zetaff` for more details. In general, these functions runs slower and slower as `N` increases.

Value

`dzipf` gives the density, `pzipf` gives the cumulative distribution function, `qzipf` gives the quantile function, and `rzipf` generates random deviates.

Author(s)

T. W. Yee

See Also

zipf, Zipfmb.

Examples

```
N <- 10; shape <- 0.5; y <- 1:N
proby <- dzipf(y, N = N, shape = shape)
## Not run:  plot(proby ~ y, type = "h", col = "blue",
   ylim = c(0, 0.2), ylab = "Probability", lwd = 2, las = 1,
   main = paste0("Zipf(N = ", N, ", shape = ", shape, ")"))
## End(Not run)
sum(proby)  # Should be 1
max(abs(cumsum(proby) - pzipf(y, N = N, shape = shape)))  # 0?
```

---

zipf *Zipf Distribution Family Function*

Description

Estimates the parameter of the Zipf distribution.

Usage

```
zipf(N = NULL, lshape = "loglink", ishape = NULL)
```

Arguments

N	Number of elements, an integer satisfying 1 < N < Inf. The default is to use the maximum value of the response. If given, N must be no less that the largest response value. If N = Inf and $s > 1$ then this is the zeta distribution (use zetaff instead).
lshape	Parameter link function applied to the (positive) shape parameter s. See Links for more choices.
ishape	Optional initial value for the parameter s. The default is to choose an initial value internally. If converge failure occurs use this argument to input a value.

Details

The probability function for a response Y is

$$P(Y = y) = y^{-s} / \sum_{i=1}^{N} i^{-s}, \quad s > 0, \quad y = 1, 2, \ldots, N,$$

where s is the exponent characterizing the distribution. The mean of Y, which are returned as the fitted values, is $\mu = H_{N,s-1}/H_{N,s}$ where $H_{n,m} = \sum_{i=1}^{n} i^{-m}$ is the nth generalized harmonic number.

Zipf's law is an experimental law which is often applied to the study of the frequency of words in a corpus of natural language utterances. It states that the frequency of any word is inversely proportional to its rank in the frequency table. For example, `"the"` and `"of"` are first two most common words, and Zipf's law states that `"the"` is twice as common as `"of"`. Many other natural phenomena conform to Zipf's law.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm` and `vgam`.

Note

Upon convergence, the `N` is stored as `@misc$N`.

Author(s)

T. W. Yee

References

pp.526– of Chapter 11 of Johnson N. L., Kemp, A. W. and Kotz S. (2005). *Univariate Discrete Distributions*, 3rd edition, Hoboken, New Jersey, USA: Wiley.

See Also

`dzipf`, `zetaff`, `simulate.vlm`.

Examples

```
zdata <- data.frame(y = 1:5, ofreq = c(63, 14, 5, 1, 2))
zfit <- vglm(y ~ 1, zipf, data = zdata, trace = TRUE, weight = ofreq)
zfit <- vglm(y ~ 1, zipf(lshape = "identitylink", ishape = 3.4), data = zdata,
            trace = TRUE, weight = ofreq, crit = "coef")
zfit@misc$N
(shape.hat <- Coef(zfit))
with(zdata, weighted.mean(y, ofreq))
fitted(zfit, matrix = FALSE)
```

Zipfmb	*The Zipf-Mandelbrot Distribution*

Description

Density, distribution function, quantile function and random generation for the Mandelbrot distribution.

Usage

```
dzipfmb(x, shape, start = 1, log = FALSE)
pzipfmb(q, shape, start = 1, lower.tail = TRUE, log.p = FALSE)
qzipfmb(p, shape, start = 1)
rzipfmb(n, shape, start = 1)
```

Arguments

`x`	vector of (non-negative integer) quantiles.
`q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of random values to return.
`shape`	vector of positive shape parameter.
`start`	integer, the minimum value of the support of the distribution.
`log, log.p`	logical; if TRUE, probabilities p are given as log(p)
`lower.tail`	logical; if TRUE (default), probabilities are P[X <= x], otherwise, P[X > x].

Details

The probability mass function of the Zipf-Mandelbrot distribution is given by

$$\Pr(Y = y; s) = \frac{s\,\Gamma(y_{min})}{\Gamma(y_{min} - s)} \cdot \frac{\Gamma(y - s)}{\Gamma(y + 1)}$$

where $0 \leq b < 1$ and the starting value start being by default 1.

Value

`dzipfmb` gives the density, `pzipfmb` gives the distribution function, `qzipfmb` gives the quantile function, and `rzipfmb` generates random deviates.

Author(s)

M. Chou, with edits by T. W. Yee.

References

Mandelbrot, B. (1961). On the theory of word frequencies and on related Markovian models of discourse. In R. Jakobson, *Structure of Language and its Mathematical Aspects*, pp. 190–219, Providence, RI, USA. American Mathematical Society.

Moreno-Sanchez, I. and Font-Clos, F. and Corral, A. (2016). Large-Scale Analysis of Zipf's Law in English Texts. *PLos ONE*, **11**(1), 1–19.

See Also

Zipf.

Examples

```
aa <- 1:10
(pp <- pzipfmb(aa, shape = 0.5, start = 1))
cumsum(dzipfmb(aa, shape = 0.5, start = 1))  # Should be same
qzipfmb(pp, shape = 0.5, start = 1) - aa  # Should be  all 0s

rdiffzeta(30, 0.5)

## Not run: x <- 1:10
plot(x, dzipfmb(x, shape = 0.5), type = "h", ylim = 0:1,
     sub = "shape=0.5", las = 1, col = "blue", ylab = "Probability",
     main = "Zipf-Mandelbrot distribution: blue=PMF; orange=CDF")
lines(x+0.1, pzipfmb(x, shape = 0.5), col = "red", lty = 3, type = "h")

## End(Not run)
```

Zipois *Zero-Inflated Poisson Distribution*

Description

Density, distribution function, quantile function and random generation for the zero-inflated and zero-deflated Poisson distribution with parameter `pstr0`.

Usage

```
dzipois(x, lambda, pstr0 = 0, log = FALSE)
pzipois(q, lambda, pstr0 = 0)
qzipois(p, lambda, pstr0 = 0)
rzipois(n, lambda, pstr0 = 0)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. Must be a single positive integer.
`lambda`	Vector of positive means.
`pstr0`	Probability of a structural zero (i.e., ignoring the Poisson distribution), called ϕ. The default value of $\phi = 0$ corresponds to the response having an ordinary Poisson distribution. If ϕ lies in (0, 1) then this is known as the zero-inflated Poisson (ZIP) distribution. This argument may be negative to allow for 0-deflation, hence its interpretation as a probability ceases.
`log`	Logical. Return the logarithm of the answer?

Details

The probability function of Y is 0 with probability ϕ, and $Poisson(\lambda)$ with probability $1-\phi$. Thus

$$P(Y = 0) = \phi + (1 - \phi)P(W = 0)$$

where W is distributed $Poisson(\lambda)$.

Value

`dzipois` gives the density, `pzipois` gives the distribution function, `qzipois` gives the quantile function, and `rzipois` generates random deviates.

Note

The argument `pstr0` is recycled to the required length, and must have values which lie in the interval $[0, 1]$.

These functions actually allow for the *zero-deflated Poisson* (ZDP) distribution. Here, `pstr0` is also permitted to lie in the interval `[-1/expm1(lambda), 0]`. The resulting probability of a zero count is *less than* the nominal Poisson value, and the use of `pstr0` to stand for the probability of a structural zero loses its meaning. When `pstr0` equals `-1/expm1(lambda)` this corresponds to the positive-Poisson distribution (e.g., see `Gaitdpois`), also called the zero-truncated Poisson or ZTP.

The zero-*modified* Poisson (ZMP) is a combination of the ZIP and ZDP and ZTP distributions. The family function

Author(s)

T. W. Yee

See Also

`zipoisson`, `Gaitdpois`, `dpois`, `rzinegbin`.

Examples

```
lambda <- 3; pstr0 <- 0.2; x <- (-1):7
(ii <- dzipois(x, lambda, pstr0 = pstr0))
max(abs(cumsum(ii) - pzipois(x, lambda, pstr0 = pstr0)))  # 0?
table(rzipois(100, lambda, pstr0 = pstr0))

table(qzipois(runif(100), lambda, pstr0))
round(dzipois(0:10, lambda, pstr0 = pstr0) * 100)  # Similar?

## Not run:  x <- 0:10
par(mfrow = c(2, 1))  # Zero-inflated Poisson
barplot(rbind(dzipois(x, lambda, pstr0 = pstr0), dpois(x, lambda)),
        beside = TRUE, col = c("blue", "orange"),
        main = paste0("ZIP(", lambda,
                      ", pstr0 = ", pstr0, ") (blue) vs",
                      " Poisson(", lambda, ") (orange)"),
        names.arg = as.character(x))

deflat.limit <- -1 / expm1(lambda)  # Zero-deflated Poisson
newpstr0 <- round(deflat.limit / 1.5, 3)
barplot(rbind(dzipois(x, lambda, pstr0 = newpstr0),
                dpois(x, lambda)),
        beside = TRUE, col = c("blue","orange"),
        main = paste0("ZDP(", lambda, ", pstr0 = ", newpstr0, ")",
                     " (blue) vs Poisson(", lambda, ") (orange)"),
        names.arg = as.character(x))
## End(Not run)
```

---

zipoisson *Zero-Inflated Poisson Distribution Family Function*

Description

Fits a zero-inflated or zero-deflated Poisson distribution by full maximum likelihood estimation.

Usage

```
zipoisson(lpstr0 = "logitlink", llambda = "loglink", type.fitted =
   c("mean", "lambda", "pobs0", "pstr0", "onempstr0"),
   ipstr0 = NULL, ilambda = NULL, gpstr0 = NULL, imethod = 1,
   ishrinkage = 0.95, probs.y = 0.35, parallel = FALSE, zero = NULL)
zipoissonff(llambda = "loglink", lonempstr0 = "logitlink",
  type.fitted = c("mean", "lambda", "pobs0", "pstr0", "onempstr0"),
  ilambda = NULL, ionempstr0 = NULL, gonempstr0 = NULL,
  imethod = 1, ishrinkage = 0.95, probs.y = 0.35, zero = "onempstr0")
```

Arguments

`lpstr0, llambda`
: Link function for the parameter ϕ and the usual λ parameter. See `Links` for more choices; see `CommonVGAMffArguments` for more information. For the zero-*deflated* model see below.

`ipstr0, ilambda`
: Optional initial values for ϕ, whose values must lie between 0 and 1. Optional initial values for λ, whose values must be positive. The defaults are to compute an initial value internally for each. If a vector then recycling is used.

`lonempstr0, ionempstr0`
: Corresponding arguments for the other parameterization. See details below.

`type.fitted`
: Character. The type of fitted value to be returned. The first choice (the expected value) is the default. The estimated probability of an observed 0 is an alternative, else the estimated probability of a structural 0, or one minus the estimated probability of a structural 0. See `CommonVGAMffArguments` and `fittedvlm` for more information.

`imethod`
: An integer with value `1` or `2` which specifies the initialization method for λ. If failure to converge occurs try another value and/or else specify a value for `ishrinkage` and/or else specify a value for `ipstr0`. See `CommonVGAMffArguments` for more information.

`ishrinkage`
: How much shrinkage is used when initializing λ. The value must be between 0 and 1 inclusive, and a value of 0 means the individual response values are used, and a value of 1 means the median or mean is used. This argument is used in conjunction with `imethod`. See `CommonVGAMffArguments` for more information.

`zero`
: Specifies which linear/additive predictors are to be modelled as intercept-only. If given, the value can be either 1 or 2, and the default is none of them. Setting `zero = 1` makes ϕ a single parameter. See `CommonVGAMffArguments` for more information.

`gpstr0, gonempstr0, probs.y`
: Details at `CommonVGAMffArguments`.

`parallel`
: Details at `CommonVGAMffArguments`, but unlikely to be practically used actually.

Details

These models are a mixture of a Poisson distribution and the value 0; it has value 0 with probability ϕ else is Poisson(λ) distributed. Thus there are two sources for zero values, and ϕ is the probability of a *structural zero*. The model for `zipoisson()` can be written

$$P(Y = 0) = \phi + (1 - \phi)\exp(-\lambda),$$

and for $y = 1, 2, \ldots$,

$$P(Y = y) = (1 - \phi)\exp(-\lambda)\lambda^y / y!.$$

Here, the parameter ϕ satisfies $0 < \phi < 1$. The mean of Y is $(1 - \phi)\lambda$ and these are returned as the fitted values, by default. The variance of Y is $(1 - \phi)\lambda(1 + \phi\lambda)$. By default, the two linear/additive predictors of `zipoisson()` are $(logit(\phi), \log(\lambda))^T$.

The **VGAM** family function `zipoissonff()` has a few changes compared to `zipoisson()`. These are: (i) the order of the linear/additive predictors is switched so the Poisson mean comes first; (ii) `onempstr0` is now 1 minus the probability of a structural 0, i.e., the probability of the parent (Poisson) component, i.e., `onempstr0` is `1-pstr0`; (iii) argument `zero` has a new default so that the `onempstr0` is intercept-only by default. Now `zipoissonff()` is generally recommended over `zipoisson()` (and definitely recommended over `yip88`). Both functions implement Fisher scoring and can handle multiple responses.

Both family functions can fit the zero-*modified* Poisson (ZMP), which is a combination of the ZIP and *zero-deflated Poisson* (ZDP); see `Zipois` for some details and the example below. The key is to set the link function to be `identitylink`. However, problems might occur when iterations get close to or go past the boundary of the parameter space, especially when there are covariates. The PMF of the ZMP is best written not as above but in terms of `onempstr0` which may be greater than unity; when using `pstr0` the above PMF is negative for non-zero values.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, `rrvglm` and `vgam`.

Warning

Numerical problems can occur, e.g., when the probability of zero is actually less than, not more than, the nominal probability of zero. For example, in the Angers and Biswas (2003) data below, replacing 182 by 1 results in nonconvergence. Half-stepping is not uncommon. If failure to converge occurs, try using combinations of `imethod`, `ishrinkage`, `ipstr0`, and/or `zipoisson(zero = 1)` if there are explanatory variables. The default for `zipoissonff()` is to model the structural zero probability as an intercept-only.

Note

This family function can be used to estimate the 0-*deflated* model, hence `pstr0` is not to be interpreted as a probability. One should set, e.g., `lpstr0 = "identitylink"`. Likewise, the functions in `Zipois` can handle the zero-deflated Poisson distribution too. Although the iterations might fall outside the parameter space, the `validparams` slot should keep them inside. A (somewhat) similar alternative for zero-deflation is to try the zero-altered Poisson model (see `zapoisson`).

The use of this **VGAM** family function with `rrvglm` can result in a so-called COZIGAM or COZIGLM. That is, a reduced-rank zero-inflated Poisson model (RR-ZIP) is a constrained zero-inflated generalized linear model. See what used to be **COZIGAM** on CRAN. A RR-ZINB model can also be fitted easily; see `zinegbinomial`. Jargon-wise, a COZIGLM might be better described as a COZIVGLM-ZIP.

Author(s)

T. W. Yee

References

Thas, O. and Rayner, J. C. W. (2005). Smooth tests for the zero-inflated Poisson distribution. *Biometrics*, **61**, 808–815.

Data: Angers, J-F. and Biswas, A. (2003). A Bayesian analysis of zero-inflated generalized Poisson model. *Computational Statistics & Data Analysis*, **42**, 37–46.

Cameron, A. C. and Trivedi, P. K. (1998). *Regression Analysis of Count Data*. Cambridge University Press: Cambridge.

M'Kendrick, A. G. (1925). Applications of mathematics to medical problems. *Proc. Edinb. Math. Soc.*, **44**, 98–130.

Yee, T. W. (2014). Reduced-rank vector generalized linear models with two linear predictors. *Computational Statistics and Data Analysis*, **71**, 889–902.

See Also

gaitdpoisson, zapoisson, Zipois, yip88, spikeplot, lpossums, rrvglm, negbinomial, zipebcom, rpois, simulate.vlm, hdeff.vglm.

Examples

```
# Example 1: simulated ZIP data
zdata <- data.frame(x2 = runif(nn <- 1000))
zdata <- transform(zdata,
           pstr01  = logitlink(-0.5 + 1*x2, inverse = TRUE),
           pstr02  = logitlink( 0.5 - 1*x2, inverse = TRUE),
           Ps01    = logitlink(-0.5       , inverse = TRUE),
           Ps02    = logitlink( 0.5       , inverse = TRUE),
           lambda1 =   loglink(-0.5 + 2*x2, inverse = TRUE),
           lambda2 =   loglink( 0.5 + 2*x2, inverse = TRUE))
zdata <- transform(zdata, y1 = rzipois(nn, lambda1, pstr0 = Ps01),
                          y2 = rzipois(nn, lambda2, pstr0 = Ps02))

with(zdata, table(y1))  # Eyeball the data
with(zdata, table(y2))
fit1 <- vglm(y1 ~ x2, zipoisson(zero = 1), zdata, crit = "coef")
fit2 <- vglm(y2 ~ x2, zipoisson(zero = 1), zdata, crit = "coef")
coef(fit1, matrix = TRUE)  # Should agree with the above values
coef(fit2, matrix = TRUE)  # Should agree with the above values

# Fit all two simultaneously, using a different parameterization:
fit12 <- vglm(cbind(y1, y2) ~ x2, zipoissonff, zdata, crit = "coef")
coef(fit12, matrix = TRUE)  # Should agree with the above values

# For the first observation compute the probability that y1 is
# due to a structural zero.
(fitted(fit1, type = "pstr0") / fitted(fit1, type = "pobs0"))[1]


# Example 2: McKendrick (1925). From 223 Indian village households
cholera <- data.frame(ncases = 0:4,  # Number of cholera cases,
                      wfreq  = c(168, 32, 16, 6, 1))  # Frequencies
fit <- vglm(ncases ~ 1, zipoisson, wei = wfreq, cholera)
coef(fit, matrix = TRUE)
with(cholera, cbind(actual = wfreq,
                    fitted = round(dzipois(ncases, Coef(fit)[2],
```

```
                                           pstr0 = Coef(fit)[1]) *
                                   sum(wfreq), digits = 2)))

# Example 3: data from Angers and Biswas (2003)
abdata <- data.frame(y = 0:7, w = c(182, 41, 12, 2, 2, 0, 0, 1))
abdata <- subset(abdata, w > 0)
fit3 <- vglm(y ~ 1, zipoisson(lpstr0 = probitlink, ipstr0 = 0.8),
             data = abdata, weight = w, trace = TRUE)
fitted(fit3, type = "pobs0")  # Estimate of P(Y = 0)
coef(fit3, matrix = TRUE)
Coef(fit3)  # Estimate of pstr0 and lambda
fitted(fit3)
with(abdata, weighted.mean(y, w))  # Compare this with fitted(fit)
summary(fit3)

# Example 4: zero-deflated (ZDP) model for intercept-only data
zdata <- transform(zdata, lambda3 = loglink(0.0, inverse = TRUE))
zdata <- transform(zdata, deflat.limit=-1/expm1(lambda3))  # Bndy
# The 'pstr0' parameter is negative and in parameter space:
# Not too near the boundary:
zdata <- transform(zdata, usepstr0 = deflat.limit / 2)
zdata <- transform(zdata,
                   y3 = rzipois(nn, lambda3, pstr0 = usepstr0))
head(zdata)
with(zdata, table(y3))  # A lot of deflation
fit4 <- vglm(y3 ~ 1, data = zdata, trace = TRUE, crit = "coef",
             zipoisson(lpstr0 = "identitylink"))
coef(fit4, matrix = TRUE)
# Check how accurate it was:
zdata[1, "usepstr0"]  # Answer
coef(fit4)[1]         # Estimate
Coef(fit4)
vcov(fit4)  # Is positive-definite

# Example 5: RR-ZIP
set.seed(123)
rrzip <- rrvglm(Alopacce ~ sm.bs(WaterCon, df = 3),
                zipoisson(zero = NULL),
                data = hspider, trace = TRUE, Index.corner = 2)
coef(rrzip, matrix = TRUE)
Coef(rrzip)
summary(rrzip)
## Not run: plotvgam(rrzip, lcol = "blue")
```

Zoabeta *The Zero/One-Inflated Beta Distribution*

Description

Density, distribution function, and random generation for the zero/one-inflated beta distribution.

Usage

```
dzoabeta(x, shape1, shape2, pobs0 = 0, pobs1 = 0, log = FALSE,
         tol = .Machine$double.eps)
pzoabeta(q, shape1, shape2, pobs0 = 0, pobs1 = 0,
         lower.tail = TRUE, log.p = FALSE, tol = .Machine$double.eps)
qzoabeta(p, shape1, shape2, pobs0 = 0, pobs1 = 0,
         lower.tail = TRUE, log.p = FALSE, tol = .Machine$double.eps)
rzoabeta(n, shape1, shape2, pobs0 = 0, pobs1 = 0,
         tol = .Machine$double.eps)
```

Arguments

`x, q, p, n`	Same as `Beta`.
`pobs0, pobs1`	vector of probabilities that 0 and 1 are observed (ω_0 and ω_1).
`shape1, shape2`	Same as `Beta`. They are called `a` and `b` in `beta` respectively.
`lower.tail, log, log.p`	Same as `Beta`.
`tol`	Numeric, tolerance for testing equality with 0 and 1.

Details

This distribution is a mixture of a discrete distribution with a continuous distribution. The cumulative distribution function of Y is

$$F(y) = (1 - \omega_0 - \omega_1)B(y) + \omega_0 \times I[0 \le y] + \omega_1 \times I[1 \le y]$$

where $B(y)$ is the cumulative distribution function of the beta distribution with the same shape parameters (`pbeta`), ω_0 is the inflated probability at 0 and ω_1 is the inflated probability at 1. The default values of ω_j mean that these functions behave like the ordinary `Beta` when only the essential arguments are inputted.

Value

`dzoabeta` gives the density, `pzoabeta` gives the distribution function, `qzoabeta` gives the quantile, and `rzoabeta` generates random deviates.

Author(s)

Xiangjie Xue and T. W. Yee

See Also

`zoabetaR`, `beta`, `betaR`, `Betabinom`.

Examples

```
## Not run:
N <- 1000; y <- rzoabeta(N, 2, 3, 0.2, 0.2)
hist(y, probability = TRUE, border = "blue", las = 1,
     main = "Blue = 0- and 1-altered; orange = ordinary beta")
sum(y == 0) / N  # Proportion of 0s
sum(y == 1) / N  # Proportion of 1s
Ngrid <- 1000
lines(seq(0, 1, length = Ngrid),
      dbeta(seq(0, 1, length = Ngrid), 2, 3), col = "orange")
lines(seq(0, 1, length = Ngrid), col = "blue",
      dzoabeta(seq(0, 1, length = Ngrid), 2 , 3, 0.2, 0.2))

## End(Not run)
```

---

zoabetaR *Zero- and One-Inflated Beta Distribution Family Function*

Description

Estimation of the shape parameters of the two-parameter beta distribution plus the probabilities of a 0 and/or a 1.

Usage

```
zoabetaR(lshape1 = "loglink", lshape2 = "loglink", lpobs0 = "logitlink",
   lpobs1 = "logitlink", ishape1 = NULL, ishape2 = NULL, trim = 0.05,
   type.fitted = c("mean", "pobs0", "pobs1", "beta.mean"),
   parallel.shape = FALSE, parallel.pobs = FALSE, zero = NULL)
```

Arguments

`lshape1, lshape2, lpobs0, lpobs1`
: Details at `CommonVGAMffArguments`. See `Links` for more choices.

`ishape1, ishape2`
: Details at `CommonVGAMffArguments`.

`trim, zero`
: Same as `betaR`. See `CommonVGAMffArguments` for information.

`parallel.shape, parallel.pobs`
: See `CommonVGAMffArguments` for more information.

`type.fitted`
: The choice `"beta.mean"` mean to return the mean of the beta distribution; the 0s and 1s are ignored. See `CommonVGAMffArguments` for more information.

Details

The standard 2-parameter beta distribution has a support on (0,1), however, many datasets have 0 and/or 1 values too. This family function handles 0s and 1s (at least one of them must be present) in the data set by modelling the probability of a 0 by a logistic regression (default link is the logit), and similarly for the probability of a 1. The remaining proportion, `1-pobs0-pobs1`, of the data comes from a standard beta distribution. This family function therefore extends `betaR`. One has $M = 3$ or $M = 4$ per response. Multiple responses are allowed.

Value

Similar to `betaR`.

Author(s)

Thomas W. Yee and Xiangjie Xue.

See Also

`Zoabeta`, `betaR`, `betaff`, `Beta`, `zipoisson`.

Examples

```
nn <- 1000; set.seed(1)
bdata <- data.frame(x2 = runif(nn))
bdata <- transform(bdata,
  pobs0 = logitlink(-2 + x2, inverse = TRUE),
  pobs1 = logitlink(-2 + x2, inverse = TRUE))
bdata <- transform(bdata,
  y1 = rzoabeta(nn, shape1 = exp(1 + x2), shape2 = exp(2 - x2),
                pobs0 = pobs0, pobs1 = pobs1))
summary(bdata)
fit1 <- vglm(y1 ~ x2, zoabetaR(parallel.pobs = TRUE),
             data = bdata, trace = TRUE)
coef(fit1, matrix = TRUE)
summary(fit1)
```

Index

∗ **Constrained additive ordination**
cao, 151
cao.control, 155
∗ **Constraint matrices**
CM.equid, 182
constraints, 207
∗ **DFBETAs**
hatvalues, 416
∗ **Hauck–Donner effect**
hdeff, 418
hdeffsev, 422
∗ **Penalized regression**
vgam, 850
∗ **Reduced-Rank Vector Generalized Linear Model**
rrvglm, 735
rrvglm-class, 738
rrvglm.control, 741
rrvglm.optim.control, 745
∗ **Reduced-rank regression**
rrvglm, 735
∗ **Vector Generalized Additive Model**
vgam, 850
vgam-class, 854
vgam.control, 857
∗ **Vector Generalized Linear Model**
vglm, 859
vglm-class, 865
vglm.control, 868
vglmff-class, 872
∗ **classes**
calibrate-methods, 143
Coef.qrrvglm-class, 187
Coef.rrvglm-class, 190
concoef-methods, 204
rrvglm-class, 738
SurvS4-class, 809
vgam-class, 854
vglm-class, 865
vglmff-class, 872
∗ **datagen**
posbernUC, 662
rcqo, 711
simulate.vlm, 761
∗ **datasets**
auuc, 57
backPain, 59
beggs, 60
bmi.nz, 131
budworm, 140
cfibrosis, 173
chest.nz, 175
chinese.nz, 176
coalminers, 183
corbet, 210
crashes, 218
deermice, 231
ducklings, 256
enzyme, 257
finney44, 299
flourbeetle, 307
gew, 389
grain.us, 400
hormone, 424
hspider, 426
Huggins89.t1, 429
hunua, 431
lakeO, 466
leukemia, 476
lirat, 492
lpossums, 532
machinists, 546
marital.nz, 553
melbmaxtemp, 561
olympics, 605
oxtemp, 612
pneumo, 646
prats, 676

prinia, 683
ruge, 746
toxop, 821
ucberk, 840
V1, 844
V2, 845
venice, 848
waitakere, 880
wine, 891
∗ **distribution**
alaplaceUC, 32
Benford, 62
Benini, 64
Betabinom, 67
Betageom, 78
Betanorm, 82
Biamhcop, 87
Biclaytoncop, 90
Bifgmcop, 94
bilogis, 101
Binom2.or, 104
Binom2.rho, 109
Binorm, 116
Binormcop, 121
Biplackett, 122
Bisa, 125
Bistudentt, 129
Bort, 134
Card, 158
Dagum, 226
dAR1, 229
dextlogF, 235
dgaitdplot, 237
dhuber, 240
Diffzeta, 242
dlogF, 250
Expectiles-Exponential, 261
Expectiles-Normal, 262
Expectiles-sc.t2, 264
Expectiles-Uniform, 265
expgeom, 272
explog, 277
exppois, 281
Extbetabinom, 284
Felix, 292
Fisk, 301
Foldnorm, 308
Frank, 312
Frechet, 314
Gaitdbinom, 319
Gaitdlog, 322
Gaitdnbinom, 327
Gaitdpois, 333
Gaitdzeta, 343
GenbetaII, 359
gengammaUC, 364
Genpois0, 366
Genpois1, 368
genray, 375
Gensh, 378
gevUC, 387
Gompertz, 392
gpdUC, 398
Gumbel-II, 409
gumbelUC, 412
Hurea, 433
Hzeta, 439
Inv.gaussian, 446
Inv.lomax, 449
Inv.paralogistic, 452
Kumar, 463
laplaceUC, 470
lgammaUC, 480
Lindley, 481
Lino, 488
Log, 501
loglapUC, 517
Lomax, 529
Makeham, 548
Maxwell, 555
Nakagami, 587
Paralogistic, 613
Pareto, 616
ParetoIV, 619
Perks, 624
PoissonPoints, 650
Polono, 651
posbernUC, 662
Posgeom, 666
Posnorm, 670
Rayleigh, 706
rcqo, 711
rdiric, 715
Rice, 724
Simplex, 759
Sinmad, 763

Skellam, 766
skewnorm, 769
Slash, 772
spikeplot, 787
Tobit, 811
Topple, 819
Triangle, 822
Trinorm, 828
Truncpareto, 836
UtilitiesVGAM, 843
Yules, 894
Zabinom, 897
Zageom, 900
Zanegbin, 903
Zapois, 908
Zeta, 913
Zibinom, 918
Zigeom, 922
Zinegbin, 925
Zipf, 933
Zipfmb, 936
Zipois, 937
Zoabeta, 943
∗ **dplot**
dgaitdplot, 237
plotvgam.control, 643
prplot, 688
∗ **hplot**
biplot-methods, 125
deplot.lmscreg, 232
dgaitdplot, 237
guplot, 414
lvplot, 538
lvplot.qrrvglm, 539
lvplot.rrvglm, 544
meplot, 562
plotdeplot.lmscreg, 632
plotdgaitd.vglm, 634
plotqrrvglm, 635
plotqtplot.lmscreg, 636
plotrcim0, 638
plotvgam, 640
plotvglm, 644
prplot, 688
qtplot.gumbel, 696
qtplot.lmscreg, 698
rlplot.gevff, 728
spikeplot, 787
trplot, 831
trplot.qrrvglm, 832
vplot.profile, 877
∗ **htest**
anova.vglm, 46
cops, 209
hdeff, 418
hdeffsev, 422
lrt.stat, 535
lrtest, 536
score.stat, 750
wald.stat, 882
∗ **manip**
iam, 441
∗ **math**
alogitlink, 33
asinlink, 55
bell, 61
cauchitlink, 162
clogloglink, 180
erf, 258
expint, 274
explink, 276
fisherzlink, 300
identitylink, 443
kendall.tau, 461
lambertW, 467
lerch, 474
logclink, 503
logitlink, 509
logitoffsetlink, 512
loglink, 523
logloglink, 525
logofflink, 527
mills.ratio, 566
multilogitlink, 581
nbcanlink, 590
ordpoisson, 608
pgamma.deriv, 629
pgamma.deriv.unscaled, 631
powerlink, 675
probitlink, 684
reciprocallink, 719
rhobitlink, 722
round2, 732
sloglink, 775
sqrtlink, 789
zeta, 914

∗ **methods**
calibrate-methods, 143
concoef-methods, 204
∗ **models**
A1A2A3, 17
AA.Aa.aa, 18
AB.Ab.aB.ab, 20
ABO, 21
acat, 22
add1.vglm, 24
AICvlm, 25
alaplace, 27
alogitlink, 33
altered, 36
amlbinomial, 37
amlexponential, 39
amlnormal, 41
amlpoisson, 44
AR1, 48
asinlink, 55
aux.posbernoulli.t, 58
benini1, 65
betabinomial, 70
betabinomialff, 73
betaff, 76
betageometric, 79
betaII, 81
betaprime, 84
betaR, 85
biamhcop, 88
biclaytoncop, 91
BICvlm, 93
bifgmcop, 95
bifgmexp, 96
bifrankcop, 98
bigumbelIexp, 99
bilogistic, 102
binom2.or, 106
binom2.rho, 111
binomialff, 113
binormal, 117
binormalcop, 119
biplackettcop, 123
bisa, 127
bistudentt, 130
borel.tanner, 132
Brat, 135
brat, 136
bratt, 138
calibrate, 141
calibrate-methods, 143
calibrate.qrrvglm, 143
calibrate.qrrvglm.control, 147
calibrate.rrvglm, 148
calibrate.rrvglm.control, 150
cao, 151
cao.control, 155
cardioid, 160
cauchitlink, 162
cauchy, 164
cdf.lmscreg, 166
cens.gumbel, 167
cens.normal, 169
cens.poisson, 171
cgo, 174
chisq, 177
clo, 178
clogloglink, 180
CM.equid, 182
Coef, 184
Coef.qrrvglm, 185
Coef.rrvglm, 189
Coef.vlm, 191
coefvgam, 192
coefvlm, 193
CommonVGAMffArguments, 194
concoef, 203
concoef-methods, 204
confintvglm, 205
constraints, 207
cops, 209
cqo, 211
cratio, 220
cumulative, 222
dagum, 228
depvar, 234
df.residual, 236
diffzeta, 243
dirichlet, 244
dirmul.old, 246
dirmultinomial, 248
double.cens.normal, 251
double.expbinomial, 253
eCDF, 256
erlang, 259
expexpff, 268

expexpff1, 270
expgeometric, 273
explink, 276
explogff, 278
exponential, 280
exppoisson, 283
extbetabinomial, 286
extlogF1, 288
familyname, 291
felix, 293
fff, 294
fill1, 296
fisherzlink, 300
fisk, 302
fittedvlm, 304
fix.crossing, 306
foldnormal, 309
formulavlm, 311
frechet, 315
freund61, 317
gaitdlog, 324
gaitdnbinomial, 329
gaitdpoisson, 337
gaitdzeta, 346
gamma1, 348
gamma2, 350
gammaff.mm, 352
gammahyperbola, 354
gammaR, 355
garma, 357
genbetaII, 360
gengamma.stacy, 362
genpoisson0, 369
genpoisson1, 372
genpoisson2, 373
genrayleigh, 376
gensh, 379
geometric, 381
get.smart, 382
get.smart.prediction, 383
gev, 384
gompertz, 393
gpd, 395
grc, 401
gumbel, 406
gumbelII, 410
has.interceptvlm, 415
hdeff, 418
hdeffsev, 422
huber2, 427
hurea, 434
hyperg, 435
hypersecant, 437
hzeta, 440
identitylink, 443
Influence, 444
inv.binomial, 445
inv.gaussianff, 448
inv.lomax, 451
inv.paralogistic, 454
is.buggy, 455
is.crossing, 457
is.parallel, 458
is.smart, 459
is.zero, 460
KLD, 462
kumar, 465
laplace, 468
latvar, 472
leipnik, 473
levy, 477
lgamma1, 478
lindley, 483
linkfun, 484
Links, 485
lino, 490
lms.bcg, 493
lms.bcn, 495
lms.yjn, 498
logclink, 503
logF, 504
logff, 506
logistic, 508
logitlink, 509
logitoffsetlink, 512
loglaplace, 513
logLik.vlm, 519
loglinb2, 520
loglinb3, 522
loglink, 523
logloglink, 525
lognormal, 526
logofflink, 527
lomax, 530
lqnorm, 533
lrt.stat, 535

makeham, 549
margeff, 551
Max, 554
maxwell, 557
mccullagh89, 558
meangaitd, 559
micmen, 564
mix2exp, 567
mix2normal, 569
mix2poisson, 571
MNSs, 573
model.framevlm, 574
model.matrixqrrvglm, 576
model.matrixvlm, 577
multilogitlink, 581
multinomial, 582
nakagami, 588
nbcanlink, 590
negbinomial, 592
negbinomial.size, 599
normal.vcm, 601
nparam.vlm, 604
Opt, 607
ordpoisson, 608
ordsup, 610
paralogistic, 614
paretoff, 617
paretoIV, 621
perks, 625
perspqrrvglm, 627
plotvgam, 640
plotvglm, 644
poisson.points, 647
poissonff, 648
posbernoulli.b, 653
posbernoulli.t, 656
posbernoulli.tb, 659
posbinomial, 664
posnegbinomial, 667
posnormal, 672
pospoisson, 674
powerlink, 675
predictqrrvglm, 678
predictvglm, 679
prentice74, 681
probitlink, 684
profilevglm, 686
propodds, 687
put.smart, 690
qrrvglm.control, 691
Qvar, 699
qvar, 703
R2latvar, 704
Rank, 705
rayleigh, 707
rec.exp1, 716
rec.normal, 717
reciprocallink, 719
residualsvglm, 720
rhobitlink, 722
riceff, 725
rigff, 727
rootogram4, 730
rrar, 733
rrvglm, 735
rrvglm.control, 741
rrvglm.optim.control, 745
s, 747
sc.studentt2, 749
score.stat, 750
seglines, 752
Select, 753
seq2binomial, 756
setup.smart, 757
simplex, 760
simulate.vlm, 761
sinmad, 764
skellam, 767
skewnormal, 770
slash, 773
sloglink, 775
sm.os, 776
sm.ps, 780
smart.expression, 783
smart.mode.is, 783
smartpred, 784
specials, 786
sqrtlink, 789
sratio, 791
step4, 794
studentt, 795
summary.drrvglm, 797
summarypvgam, 800
summaryvgam, 801
summaryvglm, 802
SURff, 805

TIC, 810
tobit, 813
Tol, 817
topple, 820
triangle, 824
trim.constraints, 826
trinormal, 829
Trunc, 835
truncweibull, 838
uninormal, 841
UtilitiesVGAM, 843
vcovvlm, 846
vgam, 850
vgam-class, 854
VGAM-package, 13
vgam.control, 857
vglm, 859
vglm.control, 868
vonmises, 875
wald.stat, 882
waldff, 884
weibull.mean, 885
weibullR, 887
weightsvglm, 890
wrapup.smart, 892
yeo.johnson, 893
yulesimon, 896
zabinomial, 898
zageometric, 901
zanegbinomial, 905
zapoisson, 909
zero, 911
zetaff, 916
zibinomial, 919
zigeometric, 923
zinegbinomial, 927
zipebcom, 930
zipf, 934
zipoisson, 939
∗ **nonlinear**
calibrate.qrrvglm, 143
calibrate.qrrvglm.control, 147
Coef.qrrvglm, 185
Coef.qrrvglm-class, 187
lvplot.qrrvglm, 539
model.matrixqrrvglm, 576
perspqrrvglm, 627
plotqrrvglm, 635
predictqrrvglm, 678
qrrvglm.control, 691
trplot.qrrvglm, 832
∗ **optimize**
calibrate.qrrvglm.control, 147
calibrate.rrvglm.control, 150
cao.control, 155
qrrvglm.control, 691
rrvglm.control, 741
rrvglm.optim.control, 745
vgam.control, 857
vglm.control, 868
∗ **package**
VGAM-package, 13
∗ **programming**
get.smart, 382
get.smart.prediction, 383
iam, 441
is.smart, 459
put.smart, 690
setup.smart, 757
smart.expression, 783
smart.mode.is, 783
smartpred, 784
UtilitiesVGAM, 843
wrapup.smart, 892
zero, 911
∗ **regression**
A1A2A3, 17
AA.Aa.aa, 18
AB.Ab.aB.ab, 20
ABO, 21
acat, 22
AICvlm, 25
alaplace, 27
alogitlink, 33
amlbinomial, 37
amlexponential, 39
amlnormal, 41
amlpoisson, 44
AR1, 48
asinlink, 55
aux.posbernoulli.t, 58
benini1, 65
betabinomial, 70
betabinomialff, 73
betaff, 76
betageometric, 79

betaII, 81
betaprime, 84
betaR, 85
biamhcop, 88
biclaytoncop, 91
BICvlm, 93
bifgmcop, 95
bifgmexp, 96
bifrankcop, 98
bigumbelIexp, 99
bilogistic, 102
binom2.or, 106
binom2.rho, 111
binomialff, 113
binormal, 117
binormalcop, 119
biplackettcop, 123
biplot-methods, 125
bisa, 127
bistudentt, 130
borel.tanner, 132
Brat, 135
brat, 136
bratt, 138
calibrate, 141
calibrate-methods, 143
calibrate.qrrvglm, 143
calibrate.qrrvglm.control, 147
calibrate.rrvglm, 148
calibrate.rrvglm.control, 150
cao, 151
cao.control, 155
cardioid, 160
cauchitlink, 162
cauchy, 164
cdf.lmscreg, 166
cens.gumbel, 167
cens.normal, 169
cens.poisson, 171
cgo, 174
chisq, 177
clo, 178
clogloglink, 180
Coef, 184
Coef.qrrvglm, 185
Coef.rrvglm, 189
Coef.vlm, 191
coefvgam, 192
coefvlm, 193
concoef, 203
concoef-methods, 204
confintvglm, 205
constraints, 207
cops, 209
cqo, 211
cratio, 220
cumulative, 222
dagum, 228
depvar, 234
df.residual, 236
diffzeta, 243
dirichlet, 244
dirmul.old, 246
dirmultinomial, 248
double.cens.normal, 251
double.expbinomial, 253
eCDF, 256
erlang, 259
expexpff, 268
expexpff1, 270
expgeometric, 273
explink, 276
explogff, 278
exponential, 280
exppoisson, 283
extbetabinomial, 286
extlogF1, 288
familyname, 291
felix, 293
fff, 294
fill1, 296
fisherzlink, 300
fisk, 302
fittedvlm, 304
fix.crossing, 306
foldnormal, 309
formulavlm, 311
frechet, 315
freund61, 317
gaitdlog, 324
gaitdnbinomial, 329
gaitdpoisson, 337
gaitdzeta, 346
gamma1, 348
gamma2, 350
gammaff.mm, 352

gammahyperbola, 354
gammaR, 355
garma, 357
genbetaII, 360
gengamma.stacy, 362
genpoisson0, 369
genpoisson1, 372
genpoisson2, 373
genrayleigh, 376
gensh, 379
geometric, 381
get.smart, 382
get.smart.prediction, 383
gev, 384
gompertz, 393
gpd, 395
grc, 401
gumbel, 406
gumbelII, 410
guplot, 414
has.interceptvlm, 415
hatvalues, 416
hdeff, 418
hdeffsev, 422
huber2, 427
hurea, 434
hyperg, 435
hypersecant, 437
hzeta, 440
identitylink, 443
inv.binomial, 445
inv.gaussianff, 448
inv.lomax, 451
inv.paralogistic, 454
is.buggy, 455
is.crossing, 457
is.parallel, 458
is.smart, 459
is.zero, 460
KLD, 462
kumar, 465
laplace, 468
latvar, 472
leipnik, 473
levy, 477
lgamma1, 478
lindley, 483
linkfun, 484
lino, 490
lms.bcg, 493
lms.bcn, 495
lms.yjn, 498
logclink, 503
logF, 504
logff, 506
logistic, 508
logitlink, 509
logitoffsetlink, 512
loglaplace, 513
logLik.vlm, 519
loglinb2, 520
loglinb3, 522
loglink, 523
logloglink, 525
lognormal, 526
logofflink, 527
lomax, 530
lqnorm, 533
lrt.stat, 535
lvplot, 538
lvplot.qrrvglm, 539
lvplot.rrvglm, 544
makeham, 549
margeff, 551
Max, 554
maxwell, 557
mccullagh89, 558
meangaitd, 559
meplot, 562
micmen, 564
mix2exp, 567
mix2normal, 569
mix2poisson, 571
MNSs, 573
multilogitlink, 581
multinomial, 582
nakagami, 588
nbcanlink, 590
negbinomial, 592
negbinomial.size, 599
normal.vcm, 601
nparam.vlm, 604
Opt, 607
ordpoisson, 608
ordsup, 610
paralogistic, 614

paretoff, 617
paretoIV, 621
perks, 625
perspqrrvglm, 627
plotdeplot.lmscreg, 632
plotdgaitd.vglm, 634
plotqrrvglm, 635
plotqtplot.lmscreg, 636
plotrcim0, 638
plotvgam, 640
plotvgam.control, 643
plotvglm, 644
poisson.points, 647
poissonff, 648
posbernoulli.b, 653
posbernoulli.t, 656
posbernoulli.tb, 659
posbinomial, 664
posnegbinomial, 667
posnormal, 672
pospoisson, 674
powerlink, 675
predictqrrvglm, 678
predictvglm, 679
prentice74, 681
probitlink, 684
profilevglm, 686
propodds, 687
prplot, 688
put.smart, 690
qrrvglm.control, 691
qtplot.gumbel, 696
qtplot.lmscreg, 698
Qvar, 699
qvar, 703
R2latvar, 704
Rank, 705
rayleigh, 707
rec.exp1, 716
rec.normal, 717
reciprocallink, 719
residualsvglm, 720
rhobitlink, 722
riceff, 725
rigff, 727
rlplot.gevff, 728
rrar, 733
rrvglm, 735
rrvglm.optim.control, 745
s, 747
sc.studentt2, 749
score.stat, 750
seglines, 752
Select, 753
seq2binomial, 756
setup.smart, 757
simplex, 760
sinmad, 764
skellam, 767
skewnormal, 770
slash, 773
sloglink, 775
sm.os, 776
sm.ps, 780
smart.expression, 783
smart.mode.is, 783
smartpred, 784
sqrtlink, 789
sratio, 791
studentt, 795
summary.drrvglm, 797
summarypvgam, 800
summaryvgam, 801
summaryvglm, 802
SURff, 805
TIC, 810
tobit, 813
Tol, 817
topple, 820
triangle, 824
trim.constraints, 826
trinormal, 829
trplot, 831
trplot.qrrvglm, 832
Trunc, 835
truncweibull, 838
uninormal, 841
UtilitiesVGAM, 843
vcovvlm, 846
vgam, 850
vgam-class, 854
VGAM-package, 13
vglm, 859
vonmises, 875
vplot.profile, 877
vsmooth.spline, 878

wald.stat, 882
waldff, 884
weibull.mean, 885
weibullR, 887
weightsvglm, 890
wrapup.smart, 892
yeo.johnson, 893
yulesimon, 896
zabinomial, 898
zageometric, 901
zanegbinomial, 905
zapoisson, 909
zero, 911
zetaff, 916
zibinomial, 919
zigeometric, 923
zinegbinomial, 927
zipebcom, 930
zipf, 934
zipoisson, 939
zoabetaR, 945
∗ **smooth**
plotvgam, 640
s, 747
sm.os, 776
sm.ps, 780
vgam, 850
vgam-class, 854
vsmooth.spline, 878
∗ **survival**
SurvS4, 807
∗ **ts**
rrar, 733
∗ **utilities**
CM.equid, 182
model.matrixqrrvglm, 576
Trunc, 835
UtilitiesVGAM, 843
.Machine, *367*
.Random.seed, *713*
[.SurvS4 *(*SurvS4*)*, 807

A1A2A3, 17, *19*, *20*, *22*, *574*
AA.Aa.aa, *18*, 18, *20*, *22*, *574*
AB.Ab.aB.ab, *18*, *19*, 20, *22*, *574*
ABO, *18–20*, 21, *574*
acat, 22, *183*, *197*, *199*, *221*, *224*, *225*, *458*, *551*, *552*, *585*, *721*, *793*, *873*
add1.glm, *24*, *25*, *794*
add1.vglm, 24, *25*, *48*, *578*, *794*, *864*
AIC, *27*, *94*, *519*, *811*
AICc,vglm-method *(*AICvlm*)*, 25
AICqrrvglm *(*AICvlm*)*, 25
AICrrvgam *(*AICvlm*)*, 25
AICrrvglm *(*AICvlm*)*, 25
AICvgam *(*AICvlm*)*, 25
AICvlm, 25, *93*, *94*, *605*, *658*, *664*, *665*, *811*, *864*
alaplace, 27
alaplace1, *29*, *38*, *41*, *43*, *46*, *289–291*, *497*, *514*, *515*, *762*, *883*
alaplace1 *(*alaplace*)*, 27
alaplace2, *403*, *470*, *762*
alaplace2 *(*alaplace*)*, 27
alaplace3, *32*, *33*, *518*
alaplace3 *(*alaplace*)*, 27
alaplaceUC, 32
alclevels *(*crashes*)*, 218
alcoff, *404*, *579*, *580*
alcoff *(*crashes*)*, 218
alogitlink, 33, *55*, *56*, *115*, *421*, *511*, *776*, *789*
altered, 36, *787*
amlbinomial, 37, *41*, *43*, *46*, *115*
amlexponential, *38*, 39, *40*, *43*, *281*, *494*
amlnormal, *30*, *38*, *41*, 41, *46*, *263*, *497*, *500*
amlpoisson, *38*, *40–43*, 44, *650*
anova, *537*
anova.glm, *46–48*
anova.vglm, *25*, 46, *236*, *519*, *536*, *537*, *751*, *864*, *883*, *884*
AR1, 48, *52–54*, *230*, *842*
AR1EIM, *49*, *50*, 52
arima.sim, *50*
array, *809*
as.character.SurvS4 *(*SurvS4*)*, 807
as.data.frame.SurvS4 *(*SurvS4*)*, 807
asinlink, *34*, *35*, 55, *115*, *421*, *511*
auuc, 57, *404*
aux.posbernoulli.t, 58, *654*

backPain, 59
backPain2, *25*, *48*, *794*
backPain2 *(*backPain*)*, 59
beggs, 60
bell, 61, *468*
bellff, *62*
Benford, 62

Benini, 64, 66
benini1, 64, 65, 65, 619
Bessel, 876
besselI, 726
Beta, 67–69, 77, 79, 85, 87, 944, 946
beta, 67, 71, 76, 78, 83, 86, 490, 505, 896, 944
Betabinom, 67, 68, 72, 75, 285, 944
betabinomial, 68, 69, 70, 74, 75, 115, 180, 249, 250, 286–288, 677, 762
betabinomialff, 68, 69, 72, 73, 77, 87, 247, 250, 287, 288, 677, 762
betaff, 72, 75, 76, 79, 80, 82, 85–87, 362, 466, 762, 946
Betageom, 78
betageometric, 77, 79, 87, 382
betaII, 77, 81, 87, 229, 304, 362, 452, 455, 532, 615, 766
Betanorm, 82
betaprime, 77, 84, 87
betaR, 77, 85, 762, 944–946
Biamhcop, 87
biamhcop, 87, 88, 88, 101, 102, 762
BIC, 94
Biclaytoncop, 90
biclaytoncop, 90, 91, 762
BICvgam (BICvlm), 93
BICvlm, 27, 93, 658, 664, 665, 811
Bifgmcop, 94
bifgmcop, 89, 94, 95, 95, 97, 99, 762
bifgmexp, 96, 96, 100
bifrankcop, 96, 98, 123, 125, 313, 762
bigamma.mckay, 351–353, 356
bigumbelIexp, 89, 97, 99
bilogis, 101, 101
bilogistic, 102, 102, 509, 762
Binom2.or, 104
binom2.or, 105, 106, 107, 108, 112, 113, 521, 523, 912, 931, 932
Binom2.rho, 109
binom2.Rho (binom2.rho), 111
binom2.rho, 107, 108, 110, 111, 521, 723
Binomial, 285, 320, 919, 921
binomial, 114, 115
binomialff, 35, 37, 38, 55, 56, 71–75, 108, 113, 113, 137, 139, 144, 145, 149, 154, 181, 212, 216, 254, 287, 288, 308, 340, 418–420, 437, 650, 654, 665, 695, 704, 714, 721, 757, 761, 762, 790, 852, 863, 899, 900, 903, 907, 910, 911, 920, 921
Binorm, 116
binormal, 90, 117, 117, 122, 131, 762, 829, 830, 842
binormalcop, 90, 119, 121, 122, 461, 762
Binormcop, 121
Biplackett, 122
biplackettcop, 123, 123
biplot,qrrvglm-method (biplot-methods), 125
biplot,rrvglm-method (biplot-methods), 125
biplot-methods, 125
biplot.rrvglm, 546, 737
biplot.rrvglm (lvplot.rrvglm), 544
Bisa, 125
bisa, 126, 127, 449
Bistudentt, 129
bistudentt, 119, 129, 130
bmi.nz, 43, 131, 494, 500, 554
borel.tanner, 132, 134, 294, 418
Bort, 134
boxcox, 894
Brat, 135, 137, 139
brat, 135, 136, 136, 137–139
bratt, 135–137, 138
bs, 289, 297, 386, 396, 456, 457, 537, 552, 748, 779, 781, 783–785, 827
budworm, 24, 140, 221, 225, 792, 793

calibrate, 141, 146, 150, 681
calibrate,Coef.qrrvglm-method (calibrate-methods), 143
calibrate,qrrvglm-method (calibrate-methods), 143
calibrate,rrvgam-method (calibrate-methods), 143
calibrate,rrvglm-method (calibrate-methods), 143
calibrate-methods, 143
calibrate.qrrvglm, 142, 143, 147–150, 154, 216, 679
calibrate.qrrvglm.control, 144, 146, 147, 151
calibrate.rrvglm, 142, 146, 148, 150, 151, 737
calibrate.rrvglm.control, 149, 150

cao, *13*, *114, 115*, *146*, 151, *155*, *158*, *181*, *213*, *216*, *436*, *487*, *537*, *596, 597*, *649, 650*, *695*, *852, 853*
cao.control, *152–154*, 155
Card, 158
cardioid, *159*, 160, *876*
cauchit, *165*
cauchitlink, *35*, *56*, *107, 108*, 162, *180, 181*, *221*, *225*, *357*, *486*, *511*, *602*, *685*, *793*
Cauchy, *163*, *165*
cauchy, *163*, 164, *762*
cauchy1, *162, 163*, *762*, *796*
cauchy1 (cauchy), 164
cbind, *207*, *754, 755*
cdf.lmscreg, 166, *494*, *497*, *500*
cennormal (cens.normal), 169
cens.gumbel, 167, *408*
cens.normal, 169, *252*, *814, 815*, *842*
cens.poisson, 171, *566*, *650*, *808*
cens.rayleigh (rayleigh), 707
cfibrosis, 173, *757*
cgo, 174
chest.nz, 175, *554*
chinese.nz, 176
chisq, 177, *364*, *762*
chisq.test, *721, 722*
Chisquare, *178*
choose, *246*, *249*
clo, 178
cloglink, *70*, *72*, *180*, *286–288*
cloglink (clogloglink), 180
clogloglink, *35*, *56*, *107, 108*, *114*, *153*, *163*, 180, *181*, *212*, *221–223*, *225*, *276*, *357*, *486*, *504*, *511*, *514*, *602*, *685*, *704, 705*, *792, 793*, *931, 932*
CM.equid, *24*, 182, *183*, *199*, *221*, *225*, *585*, *744*, *793*
CM.free, *183*, *799*
CM.free (CM.equid), 182
CM.ones, *183*, *225*
CM.ones (CM.equid), 182
CM.qlogis, *183*
CM.qlogis (CM.equid), 182
CM.qnorm, *183*, *199*, *223*, *742–744*
CM.qnorm (CM.equid), 182
CM.symm0, *14*, *744*
CM.symm0 (CM.equid), 182
CM.symm1, *183*, *223*
CM.symm1 (CM.equid), 182
coalminers, *108*, *113*, 183
Coef, *153*, 184, *192*, *472*, *538*, *554*, *607*, *817*, *831*
coef, *13*, *184, 185*, *191–194*, *204, 205*, *210*
coef,vgam-method (coefvgam), 192
Coef.qrrvglm, *147, 148*, *185*, 185, *186*, *188*, *216*, *540*, *542, 543*, *627*, *629*, *678*, *695*
Coef.qrrvglm-class, 187
Coef.rrvglm, *151*, *185*, 189, *191*, *546*, *737*
Coef.rrvglm-class, 190
Coef.vlm, *185*, 191
coefficients,vgam-method (coefvgam), 192
coefvgam, 192, *194*
coefvlm, *192, 193*, 193, *210*, *864*
colMeans, *660*
colnames, *339*, *403*
CommonVGAMffArguments, *16*, *19*, *21*, *23, 24*, *28*, *30*, *37*, *40*, *42*, *44*, *49*, *65, 66*, *70*, *73*, *76*, *79*, *81*, *84*, *86*, *89*, *91*, *95*, *98*, *103*, *106*, *111*, *113, 114*, *118, 119*, *124*, *127, 128*, *130*, *132*, *135*, *152*, *160*, *164*, *167, 168*, *170, 171*, *177*, *183*, 194, *208*, *212*, *220–223*, *225*, *228*, *244*, *246*, *248*, *251*, *253, 254*, *260*, *268, 269*, *271*, *273*, *278*, *280*, *283*, *286*, *289, 290*, *293*, *295*, *303*, *304*, *309, 310*, *315*, *317*, *325, 326*, *331*, *333*, *339, 340*, *343*, *347–352*, *355*, *360*, *362, 363*, *370*, *372*, *374*, *377*, *379–381*, *385*, *387*, *393*, *395*, *396*, *406*, *411*, *428*, *434*, *440*, *445*, *448*, *451*, *454*, *460*, *465*, *469*, *479*, *483*, *490*, *493*, *495*, *497*, *499*, *505*, *506*, *508, 509*, *514*, *520*, *522*, *526*, *527*, *531*, *533*, *550*, *557, 558*, *564*, *567*, *569*, *571*, *582–585*, *589*, *593–595*, *597*, *599*, *601*, *609*, *614*, *617*, *625*, *649*, *653*, *656*, *659*, *662*, *664*, *668, 669*, *672*, *674*, *682*, *708*, *718*, *725*, *749*, *756*, *760*, *765*, *768*, *770*, *774*, *792, 793*, *795*, *805*, *813–815*, *820*, *830*, *838*, *841*, *844*, *860, 861*, *875*, *884*, *886, 887*, *896*, *898–900*, *902, 903*, *905–907*, *910–912*, *917*, *920*, *924*, *927–932*,

940, *945*
concoef, *153*, *203*, 203
concoef,cao-method *(*concoef-methods*)*, 204
concoef,Coef.cao-method *(*concoef-methods*)*, 204
concoef,Coef.qrrvglm-method *(*concoef-methods*)*, 204
concoef,Coef.rrvglm-method *(*concoef-methods*)*, 204
concoef,qrrvglm-method *(*concoef-methods*)*, 204
concoef,rrvglm-method *(*concoef-methods*)*, 204
concoef-method *(*concoef-methods*)*, 204
concoef-methods, 204
confint, *144*, *205, 206*
confint.default, *205*
confint.lm, *206*
confintrrvglm *(*confintvglm*)*, 205
confintvgam *(*confintvglm*)*, 205
confintvglm, 205, *421*, *536*, *611*, *686, 687*, *804*, *848*, *864*, *877*
constraints, 207, *222, 223*, *225*, *339*, *458*, *460*, *827*, *861*, *912*
constraints.vlm, *578*, *864*
cops, 209
cops,vglm-method *(*cops*)*, 209
copsvglm *(*cops*)*, 209
cor, *461*
corbet, 210, *669*
coxph, *419*
cqo, *13*, *16*, *114, 115*, *146*, *152–154*, *174, 175*, *179*, *181*, *187, 188*, 211, *350*, *403, 404*, *427*, *436*, *487*, *537*, *543*, *576*, *593, 594*, *596, 597*, *629*, *636*, *649, 650*, *679*, *691*, *695*, *713, 714*, *736, 737*, *744*, *785*, *818*, *834*
crashbc *(*crashes*)*, 218
crashes, 218
crashf *(*crashes*)*, 218
crashi, *404*, *580*, *737*
crashi *(*crashes*)*, 218
crashmc *(*crashes*)*, 218
crashp *(*crashes*)*, 218
crashtr *(*crashes*)*, 218
cratio, *24*, *183*, *197*, *199*, 220, *224, 225*, *458*, *551, 552*, *585*, *721*, *792, 793*, *873*
cumsum, *563*
cumulative, *23, 24*, *93*, *182, 183*, *197*, *199*, *205*, *209*, *220, 221*, 222, *292*, *418*, *458*, *509*, *551, 552*, *583–585*, *611*, *646*, *687–689*, *704, 705*, *721*, *790*, *792, 793*, *873*, *884*
cut, *608*, *711*, *713, 714*

Dagum, 226, *229*
dagum, *82*, *226, 227*, 228, *304*, *362*, *452*, *455*, *532*, *615*, *762*, *766*
dalap, *235*, *290, 291*, *518*
dalap *(*alaplaceUC*)*, 32
dAR1, *50*, 229
data.frame, *574, 575*
dbenf *(*Benford*)*, 62
dbenini *(*Benini*)*, 64
dbetabinom *(*Betabinom*)*, 67
dbetageom *(*Betageom*)*, 78
dbetanorm *(*Betanorm*)*, 82
dbiamhcop *(*Biamhcop*)*, 87
dbiclaytoncop, *92*
dbiclaytoncop *(*Biclaytoncop*)*, 90
dbifgmcop *(*Bifgmcop*)*, 94
dbifrankcop *(*Frank*)*, 312
dbilogis *(*bilogis*)*, 101
dbinom, *68*, *321*, *897*, *900*
dbinom2.or *(*Binom2.or*)*, 104
dbinom2.rho *(*Binom2.rho*)*, 109
dbinorm *(*Binorm*)*, 116
dbinormcop *(*Binormcop*)*, 121
dbiplackcop *(*Biplackett*)*, 122
dbisa *(*Bisa*)*, 125
dbistudentt, *131*
dbistudentt *(*Bistudentt*)*, 129
dbort *(*Bort*)*, 134
dcard *(*Card*)*, 158
ddagum *(*Dagum*)*, 226
ddiffzeta *(*Diffzeta*)*, 242
deermice, *59*, 231, *654*, *658*, *661*
deexp, *41*, *263*, *267*
deexp *(*Expectiles-Exponential*)*, 261
deflated *(*altered*)*, 36
denorm, *38*, *43*, *262*, *267*, *497*
denorm *(*Expectiles-Normal*)*, 262
deparse1, *789*
deplot.lmscreg, *167*, 232, *494*, *497*, *500*, *633*, *699*
depvar, *185*, 234

deriv3, *23, 24*
deunif, *261–263*
deunif *(*Expectiles-Uniform*)*, 265
deviance, *236*
dexp, *262*
dexpgeom, *274*
dexpgeom *(*expgeom*)*, 272
dexplog, *279*
dexplog *(*explog*)*, 277
dexppois, *284*
dexppois *(*exppois*)*, 281
dextbetabinom *(*Extbetabinom*)*, 284
dextlogF, *32, 33*, 235, *251*, *291*
df.residual, 236
df.residual_vlm *(*df.residual*)*, 236
dfbeta *(*hatvalues*)*, 416
dfbetavlm *(*hatvalues*)*, 416
dfelix, *294*
dfelix *(*Felix*)*, 292
dfisk *(*Fisk*)*, 301
dfoldnorm *(*Foldnorm*)*, 308
dfrechet *(*Frechet*)*, 314
dgaitdbinom *(*Gaitdbinom*)*, 319
dgaitdlog, *239*
dgaitdlog *(*Gaitdlog*)*, 322
dgaitdnbinom, *240*
dgaitdnbinom *(*Gaitdnbinom*)*, 327
dgaitdplot, 237, *320*, *323*, *333*, *336*, *345*, *560, 561*, *634*, *789*
dgaitdpois, *239*
dgaitdpois *(*Gaitdpois*)*, 333
dgaitdzeta *(*Gaitdzeta*)*, 343
dgamma, *349*
dgenbetaII, *362*
dgenbetaII *(*GenbetaII*)*, 359
dgengamma.stacy *(*gengammaUC*)*, 364
dgenpois0 *(*Genpois0*)*, 366
dgenpois1 *(*Genpois1*)*, 368
dgenpois2 *(*Genpois1*)*, 368
dgenray, *377*
dgenray *(*genray*)*, 375
dgensh *(*Gensh*)*, 378
dgeom, *900*, *903*, *922, 923*
dgev, *385*
dgev *(*gevUC*)*, 387
dgompertz, *394*, *413*
dgompertz *(*Gompertz*)*, 392
dgpd, *395*, *397*
dgpd *(*gpdUC*)*, 398
dgumbel, *392*, *410*
dgumbel *(*gumbelUC*)*, 412
dgumbelII, *408*, *411*
dgumbelII *(*Gumbel-II*)*, 409
dhuber, 240
dhurea, *435*
dhurea *(*Hurea*)*, 433
dhyper, *436, 437*
dhzeta, *440*
dhzeta *(*Hzeta*)*, 439
Diffzeta, 242, *244*
diffzeta, *242, 243*, 243, *917*
digamma, *682*
dimm *(*UtilitiesVGAM*)*, 843
dinv.gaussian *(*Inv.gaussian*)*, 446
dinv.lomax *(*Inv.lomax*)*, 449
dinv.paralogistic *(*Inv.paralogistic*)*, 452
dirichlet, 244, *247*, *250*, *256*, *442*, *585*, *716*, *761, 762*
dirmul.old, 246, *250*
dirmultinomial, *72*, *75*, *245*, *247*, 248, *288*, *585*
dkumar, *466*
dkumar *(*Kumar*)*, 463
dlaplace *(*laplaceUC*)*, 470
dlgamma *(*lgammaUC*)*, 480
dlind, *484*
dlind *(*Lindley*)*, 481
dlino *(*Lino*)*, 488
dlog, *322, 323*
dlog *(*Log*)*, 501
dlogF, 250, *505*
dlogis, *508*
dloglap, *515*
dloglap *(*loglapUC*)*, 517
dlomax *(*Lomax*)*, 529
dmakeham, *392*, *549, 550*
dmakeham *(*Makeham*)*, 548
dmaxwell *(*Maxwell*)*, 555
dnaka *(*Nakagami*)*, 587
dnbinom, *328*, *904*, *907*, *926*
dnorm, *263*, *311*, *672*, *842*
do.call, *239*
double.cens.normal, *170*, 251, *718*, *815*, *842*
double.expbinomial, *115*, 253, *822*

dparalogistic (Paralogistic), 613
dpareto (Pareto), 616
dparetoI (ParetoIV), 619
dparetoII (ParetoIV), 619
dparetoIII (ParetoIV), 619
dparetoIV (ParetoIV), 619
dperks, *626*
dperks (Perks), 624
dpois, *145*, *336*, *367*, *371*, *572*, *651*, *767, 768*, *938*
dpois.points (PoissonPoints), 650
dpolono (Polono), 651
dposbern, *654*, *658*
dposbern (posbernUC), 662
dposgeom (Posgeom), 666
dposnorm (Posnorm), 670
drayleigh (Rayleigh), 706
drice, *726*
drice (Rice), 724
drop1.glm, *24, 25*, *794*
drop1.vglm, *27*, *48*, *578*, *794*, *827*, *864*
drop1.vglm (add1.vglm), 24
dsc.t2, *267*, *749, 750*
dsc.t2 (Expectiles-sc.t2), 264
dsimplex, *761*
dsimplex (Simplex), 759
dsinmad (Sinmad), 763
dskellam, *768*
dskellam (Skellam), 766
dskewnorm (skewnorm), 769
dslash (Slash), 772
dt, *129*, *265*, *508*
dtobit, *814*
dtobit (Tobit), 811
dtopple (Topple), 819
dtriangle (Triangle), 822
dtrinorm (Trinorm), 828
dtruncpareto (Truncpareto), 836
ducklings, 256
dunif, *266, 267*
dweibull, *838, 839*, *886*, *889*
dyules, *896*
dyules (Yules), 894
dzabinom, *900*
dzabinom (Zabinom), 897
dzageom, *903*
dzageom (Zageom), 900
dzanegbin (Zanegbin), 903
dzapois (Zapois), 908
dzeta, *344, 345*
dzeta (Zeta), 913
dzibinom (Zibinom), 918
dzigeom (Zigeom), 922
dzinegbin (Zinegbin), 925
dzipf, *935*
dzipf (Zipf), 933
dzipfmb (Zipfmb), 936
dzipois, *909*
dzipois (Zipois), 937
dzoabeta, *77*
dzoabeta (Zoabeta), 943
dzoibetabinom (Betabinom), 67

eCDF, 256, *291*, *497*
edhuber (dhuber), 240
eexp (Expectiles-Exponential), 261
enorm (Expectiles-Normal), 262
enzyme, 257, *565*
erf, 258
erfc (erf), 258
erlang, 259, *418*, *762*
eunif (Expectiles-Uniform), 265
exp, *275*, *468*, *503*
Expectiles-Exponential, 261
Expectiles-Normal, 262
Expectiles-sc.t2, 264
Expectiles-Uniform, 265
expexpff, 268, *270, 271*, *351*, *356*, *886*, *889*
expexpff1, *269*, 270
expexpint (expint), 274
expgeom, 272
expgeometric, *272*, 273, *281*, *382*
expint, 274
explink, 276, *524*, *700, 701*, *703*, *841*
explog, 277
explogff, *277, 278*, 278, *281*, *507*
expm1, *502, 503*
Exponential, *281*, *399*
exponential, *40, 41*, *260*, *272*, *274*, *278, 279*, 280, *284*, *318*, *355*, *470*, *568*, *717*, *721*, *762*
exppois, 281
exppoisson, *282*, 283
Extbetabinom, *69*, 284, *288*
extbetabinomial, *71, 72*, *74, 75*, *285*, 286, *528*

extlogF1, *29, 30*, *33*, *38*, *41*, *43*, *46*, *235*, *257*, 288, *306*, *457*, *497*, *505*
extlogitlink, *76, 77*, *161*, *387*, *438*, *486*
extlogitlink (logitlink), 509
extractAIC.vglm, *25*, *27*, *794*

familyname, 291
FDist, *295*
Felix, 292
felix, *133*, *292, 293*, 293, *418*
fff, 294
fill1, *224*, *231*, 296, *584, 585*, *661*, *755*, *863, 864*, *869, 870*
finney44, 299
fisherz, *723*
fisherzlink, *113*, 300, *486*, *602*, *723*
Fisk, 301, *304*
fisk, *82*, *229*, *302*, 302, *361, 362*, *452*, *455*, *532*, *615*, *762*, *766*
fitted, *305*
fitted.values.vlm (fittedvlm), 304
fittedvlm, *198*, 304, *679*, *864*, *898*, *902*, *905*, *910*, *920*, *924*, *927*, *929*, *940*
fix.crossing, *289*, *291*, 306, *457*
flourbeetle, 307
Foldnorm, 308
foldnormal, *308, 309*, 309, *771*, *842*
foldsqrtlink, *486*
foldsqrtlink (sqrtlink), 789
format.SurvS4 (SurvS4), 807
formula, *311, 312*, *537*, *754, 755*
formula.vlm (formulavlm), 311
formulavlm, 311, *416*
Frank, 312
Frechet, 314
frechet, *314, 315*, 315, *387*
freund61, *281*, 317

Gaitdbinom, 319, *329*, *336*, *898*, *919*
Gaitdlog, *321*, 322, *325, 326*, *329*, *336*, *502*
gaitdlog, *37*, *239*, *323*, 324, *333*, *339*, *343*, *348*, *391*, *507*, *836*
Gaitdnbinom, *237*, *240*, *321*, 327, *333*, *336*, *904*, *907*
gaitdnbinomial, *329*, 329, *343*, *391*, *462, 463*, *597*, *669*, *907*, *929*
Gaitdpois, *37*, *237, 238*, *240*, *320, 321*, *323*, *326*, *328, 329*, *331*, 333, *340, 341*, *343–345*, *348*, *561*, *585*, *675*, *787*, *909–911*, *938*
gaitdpoisson, *37*, *238–240*, *325, 326*, *330–333*, *336*, 337, *339*, *346–348*, *391*, *462, 463*, *561*, *582*, *585*, *634*, *650*, *675*, *787–789*, *835, 836*, *911*, *942*
Gaitdzeta, *321*, *323*, *329*, *336*, 343, *347, 348*
gaitdzeta, *37*, *326*, *333*, *343*, *345*, 346, *391*, *730*, *836*, *917*
gam, *852*, *858*, *863*
gamma, *260*, *349, 350*, *352*, *355*, *915, 916*
gamma1, 348, *351*, *356*, *364*, *479*, *762*
gamma2, *154*, *212*, *216*, 350, *356*, *364*, *589*, *693*, *695*, *712*, *714*, *762*
GammaDist, *351*
gammaff, *353*
gammaff.mm, *349*, *351*, 352, *356*
gammahyperbola, 354
gammaR, *260*, *269*, *349*, *351*, 355, *484*, *762*
garma, 357
gaussian, *842*
gaussianff (uninormal), 841
GenbetaII, 359
genbetaII, *77*, *82*, *87*, *227*, *229*, *302–304*, *359, 360*, 360, *450–453*, *455*, *491*, *530–532*, *614, 615*, *764–766*
gengamma.stacy, 362, *365*, *479*, *683*, *762*
gengammaUC, 364
Genpois0, 366, *368, 369*, *371*
Genpois1, *366, 367*, 368, *373*
Genpois2, *375*
Genpois2 (Genpois1), 368
genpoisson0, *366, 367*, 369, *370*, *372–375*, *596, 597*, *650*
genpoisson1, *370, 371*, 372, *375*, *596, 597*, *650*
genpoisson2, *197*, *370, 371*, *373*, 373, *596, 597*, *650*
genray, 375
genrayleigh, *375, 376*, 376, *709*
Gensh, 378, *380*
gensh, *379*, 379, *438*, *509*
Geometric, *382*
geometric, *79, 80*, *272*, *274*, 381, *762*, *903*, *925*
get.smart, 382, *383, 384*, *690*, *783*, *785*
get.smart.prediction, *383*, 383, *785*

gev, *169*, *316*, 384, *388*, *397*, *407, 408*, *411*, *413*, *415*, *849*, *886*, *888, 889*
gevff, *388*, *408*, *443*, *729*
gevff *(*gev*)*, 384
gevUC, 387
gew, 389, *806*
glm, *13*, *16*, *194*, *383*, *487*, *578*, *597*, *731*, *758*, *784*, *842*, *852*, *860*, *862, 863*, *874*, *891*
goffset, *326*, *333*, *343*, *348*, 390, *836*
Gompertz, 392
gompertz, *392*, 393, *550*, *762*
gpd, *281*, *387*, 395, *399*, *563*, *619*, *623*, *849*
gpdUC, 398
grain.us, 400, *734*
grc, *61*, *219*, 401, *606*, *737*
grep, *196*
gumbel, *168, 169*, *387*, 406, *411*, *413*, *415*, *697, 698*
Gumbel-II, 409
gumbelff, *169*, *387*, *413*, *415*, *697*
gumbelff *(*gumbel*)*, 406
gumbelII, *409, 410*, 410, *762*, *886*, *889*
gumbelUC, 412
guplot, *169*, *387*, *408*, 414, *849*

has.intercept *(*has.interceptvlm*)*, 415
has.interceptvlm, *312*, 415
hatplot *(*hatvalues*)*, 416
hatvalues, 416, *722*
hatvaluesvlm, *864*
hatvaluesvlm *(*hatvalues*)*, 416
hdeff, *14*, *16*, *35*, *56*, *210*, *342*, 418, *420*, *422, 423*, *525*, *535, 536*, *751*, *753*, *776*, *790*, *882–884*
hdeff.vglm, *114, 115*, *225*, *650*, *804*, *842*, *847, 848*, *864*, *942*
hdeffsev, *420, 421*, 422, *752, 753*, *884*
hist, *568*, *572*, *788*
hormone, 424
hspider, *216*, 426
huber1 *(*huber2*)*, 427
huber2, *241*, 427, *796*, *842*
Huggins89.t1, 429, *658*, *661*
Huggins89table1, *658*, *661*
Huggins89table1 *(*Huggins89.t1*)*, 429
hunua, 431, *523*, *881*
Hurea, 433
hurea, *433, 434*, 434, *435*
hyperg, 435
hypersecant, *251*, *379, 380*, 437
hypersecant01 *(*hypersecant*)*, 437
Hzeta, 439, *441*
hzeta, *439*, 440, *762*, *917*

I, *747*
iam, 441
identity, *602*, *649*, *726*
identitylink, *198*, 443, *486*, *602*, *611*, *720*, *941*
inflated, *787*
inflated *(*altered*)*, 36
Influence, 444
influence.measures, *418*
Influence.vglm, *864*
integrate, *433*, *652*, *772, 773*
interleave.VGAM *(*UtilitiesVGAM*)*, 843
inv.binomial, 445, *597*, *650*
Inv.gaussian, 446, *449*
inv.gaussianff, *128*, *447*, 448, *885*
Inv.lomax, 449
inv.lomax, *82*, *229*, *304*, *362*, *450*, 451, *452*, *455*, *532*, *615*, *762*, *766*
Inv.paralogistic, 452, *455*
inv.paralogistic, *82*, *229*, *304*, *362*, *452, 453*, 454, *532*, *615*, *762*, *766*
iris, *585*
is.altered *(*altered*)*, 36
is.buggy, 455, *748*, *852, 853*
is.crossing, *289*, *291*, *306*, 457
is.deflated *(*altered*)*, 36
is.inflated *(*altered*)*, 36
is.na.SurvS4 *(*SurvS4*)*, 807
is.parallel, *208*, 458
is.smart, 459, *779*, *781*
is.SurvS4 *(*SurvS4*)*, 807
is.truncated *(*altered*)*, 36
is.zero, *208*, 460

kendall.tau, *92*, *120*, 461
KLD, *333*, *343*, 462
KLDvglm *(*KLD*)*, 462
Kumar, 463
kumar, *77*, *87*, *464*, 465, *762*

lakeO, 466
lambertW, 467, *524*, *548, 549*
laplace, *30*, *281*, *428*, 468, *471*

laplaceUC, 470
latvar, *154*, *472*, 472, *539*
lbeta, *68*
lcalogitlink, *34*, *55*
lcalogitlink *(*alogitlink*)*, 33
lcsloglink, *789*
lcsloglink *(*sloglink*)*, 775
legend, *752*
leipnik, 473, *559*
lerch, 474, *916*
leukemia, 476, *808*
levy, 477
lfactorial, *367*
lgamma, *436*, *479*, *683*
lgamma1, *349*, 478, *480*, *481*, *762*
lgamma3, *481*, *682*, *683*, *762*
lgamma3 *(*lgamma1*)*, 478
lgammaUC, 480
Lindley, 481
lindley, *349*, *418*, *482*, 483, *762*
linkfun, 484, *485*, *487*
linkfunvlm, *864*
linkfunvlm *(*linkfun*)*, 484
Links, *16*, *17*, *19–22*, *28*, *34*, *35*, *49*, *55*, *56*, *65*, *70*, *73*, *79*, *81*, *84*, *86*, *88*, *91*, *95*, *97*, *98*, *100*, *103*, *106*, *111*, *113*, *115*, *118*, *119*, *124*, *127*, *130*, *132*, *160*, *162*, *164*, *168*, *169*, *171*, *172*, *180*, *181*, *195*, *201*, *220*, *222*, *225*, *228*, *244*, *246*, *248*, *251*, *253*, *260*, *268*, *270*, *273*, *276*, *277*, *280*, *283*, *286*, *289*, *293*, *295*, *300*, *301*, *303*, *309*, *315*, *317*, *325*, *339*, *347*, *349*, *350*, *352*, *354*, *355*, *360*, *362*, *370*, *372*, *374*, *377*, *379*, *381*, *384*, *393*, *395*, *406*, *410*, *428*, *436*, *437*, *440*, *443–445*, *448*, *451*, *454*, *465*, *469*, *473*, *477*, *478*, *483*, 485, *490*, *495*, *503–506*, *508*, *510–514*, *523–528*, *531*, *533*, *549*, *557*, *558*, *564*, *567*, *569*, *571*, *573*, *581*, *582*, *588*, *590*, *591*, *593*, *614*, *617*, *622*, *625*, *647*, *649*, *650*, *667*, *672*, *674–676*, *682*, *684*, *685*, *708*, *716*, *718*, *719*, *723*, *725*, *727*, *749*, *756*, *760*, *765*, *768*, *770*, *774–776*, *789*, *790*, *792*, *795*, *813*, *824*, *830*, *838*, *841*, *875*, *886*, *887*, *896*, *898*, *902*, *905*, *909*, *917*, *920*, *924*, *927*, *930*, *934*, *940*, *945*
Lino, 488, *491*
lino, *362*, *489*, 490, *762*
lirat, *72*, *75*, *288*, 492
list, *861*
lm, *16*, *42*, *194*, *236*, *383*, *384*, *578*, *602*, *758*, *784*
lm.influence, *444*
lms.bcg, *41*, *167*, *233*, 493, *497*, *500*, *699*
lms.bcn, *30*, *43*, *167*, *233*, *257*, *263*, *289*, *291*, *306*, *457*, *493*, *494*, 495, *499*, *500*, *561*, *699*
lms.yjn, *167*, *232*, *233*, *494*, *497*, 498, *637*, *698*, *699*, *894*
lms.yjn2 *(*lms.yjn*)*, 498
Log, 501, *507*
log, *275*, *468*, *503*, *507*, *524*
log10, *307*
log1mexp, 502
log1p, *502*, *503*
log1pexp *(*log1mexp*)*, 502
log1plink, *70*, *72*, *527*, *528*
log1plink *(*logofflink*)*, 527
logclink, *486*, 503, *524*
loge, *487*
logF, *251*, *291*, 504
logff, *63*, *326*, *501*, *502*, *505*, 506, *669*, *762*
logffMlink, *325*, *326*, *391*
Logistic, *379*, *511*
logistic, *104*, *379*, *380*, 508, *510*, *762*, *796*
logistic1, *225*, *508*, *511*, *762*
logistic1 *(*logistic*)*, 508
logit, *487*
logitlaplace1, *515*
logitlaplace1 *(*loglaplace*)*, 513
logitlink, *34*, *35*, *56*, *108*, *114*, *153*, *163*, *180*, *181*, *198*, *212*, *221*, *225*, *289*, *301*, *340*, *357*, *371*, *444*, *486*, *487*, *506*, *509*, 509, *512–514*, *524*, *581*, *582*, *602*, *611*, *685*, *704*, *705*, *793*
logitoffsetlink, *181*, *511*, 512
loglaplace, 513
loglaplace1, *518*
loglaplace1 *(*loglaplace*)*, 513
loglapUC, 517
logLik, *537*
logLik.vlm, 519
loglinb2, *108*, *113*, 520, *523*

loglinb3, *108*, *521*, 522
loglink, *163*, *198*, *277*, *289*, *357*, *444*, *486*, *487*, *491*, *504*, *507*, *511*, *514*, 523, *525*, *526*, *528*, *552*, *591*, *602*, *649*, *650*, *676*, *775*, *776*, *820*, *886*
loglog, *441*, *525*, *795*, *917*
loglog (logloglink), 525
logloglink, *486*, *504*, *524*, 525, *593*, *602*
loglogloglink, *486*
loglogloglink (logloglink), 525
logneglink (loglink), 523
Lognormal, *527*, *651*
lognormal, *364*, 526, *652*, *762*, *886*, *889*
logofflink, *384*, *395*, *486*, *504*, *507*, *524*, *526*, 527, *559*, *602*
Lomax, 529, *532*
lomax, *82*, *229*, *304*, *362*, *452*, *455*, *529*, *530*, 530, *615*, *762*, *766*
lpossums, 532, *942*
lqnorm, 533
lrt.stat, *421*, *422*, 535, *687*, *751*, *877*, *884*
lrt.stat.vlm, *48*, *206*, *420*, *537*, *803*, *804*, *847*, *848*, *864*
lrtest, *48*, *536*, 536
lrtest_vglm, *864*
lrtest_vglm (lrtest), 536
lv, *472*
lv (latvar), 472
lvplot, *125*, *153*, *215*, *472*, 538, *543*, *546*, *832*
lvplot.qrrvglm, *186*, *187*, *216*, *539*, 539, *628*, *629*, *635*, *636*, *695*
lvplot.rrvglm, 544, *546*, *737*, *741*

machinists, 546
magic, *13*, *777–779*, *781*, *853*, *858*
Makeham, 548
makeham, *394*, *548*, *549*, 549, *762*
mapply, *367*
margeff, *24*, *221*, *225*, 551, *585*, *597*, *650*, *669*, *793*, *815*
marital.nz, 553
match.call, *783*
Math.SurvS4 (SurvS4), 807
matrix, *105*, *110*, *199*, *260*, *335*, *352*, *419*, *599*, *809*, *814*
Max, *153*, *154*, 554, *608*, *818*
Maxwell, 555, *558*, *651*
maxwell, *556*, 557, *647*, *648*, *707*, *709*, *886*, *889*
mccullagh89, *474*, 558
meangaitd, *240*, *333*, *343*, 559
median, *470*
medpolish, *403*, *404*
melbmaxtemp, 561
meplot, *397*, 562
methods, *797*, *800–802*
micmen, *258*, 564, *861*
mills.ratio, *171*, *172*, 566, *815*
mills.ratio2 (mills.ratio), 566
mix2exp, *281*, 567
mix2normal, 569, *572*, *842*
mix2poisson, *568*, *570*, 571, *650*
MNSs, *18–20*, *22*, 573
model.frame, *575*
model.framevlm, 574, *577*, *578*
model.matrix, *234*, *577*, *578*
model.matrix.default, *152*, *861*
model.matrixqrrvglm, *216*, 576, *847*, *848*
model.matrixvlm, *575*, *576*, 577, *827*
moffset, *404*, *579*, 579, *640*, *710*
multilogitlink, *200*, *201*, *240*, *333*, *343*, *484*, *485*, *511*, 581, *585*, *602*
multinom, *584*
Multinomial, *585*
multinomial, *14*, *22–24*, *137*, *149*, *182*, *183*, *199*, *201*, *220–222*, *224*, *225*, *245*, *247*, *249*, *250*, *297*, *321*, *323*, *329*, *333*, *336*, *338*, *343*, *345*, *404*, *442*, *458*, *544*, *551*, *552*, *581*, *582*, 582, *687*, *721*, *792*, *793*

Nakagami, 587
nakagami, *587*, *588*, 588
names, *743*
nbcanlink, *590*, 590, *596*, *597*, *600*
nef.hs (hypersecant), 437
NegBinomial, *330*, *597*
negbinomial, *154*, *195*, *198*, *212*, *216*, *331*, *333*, *368*, *371*, *373*, *375*, *381*, *382*, *403*, *446*, *525*, *547*, *551*, *552*, *590*, *591*, 592, *599*, *600*, *649*, *650*, *652*, *668*, *669*, *693*, *695*, *712*, *714*, *737*, *762*, *906*, *907*, *926–929*, *942*
negbinomial.size, *590*, *591*, *597*, 599, *762*
negidentitylink, *486*
negidentitylink (identitylink), 443
negloglink, *351*, *356*, *486*, *593*, *595*, *622*
negloglink (loglink), 523

negreciprocallink, *486*
negreciprocallink *(*reciprocallink*)*, 719
nobs.vlm, *864*
Normal, *83*, *378*, *566*, *570*, *685*, *895*, *913*, *933*
normal.vcm, *200*, *420*, *582*, 601, *842*
nparam *(*nparam.vlm*)*, 604
nparam.vlm, 604
npred.vlm, *864*
ns, *297*, *386*, *396*, *456, 457*, *537*, *748*, *784, 785*

Oalog, *323*
oalog, *325, 326*, *507*
Oazeta, *345*, *914*
oazeta, *347, 348*, *916, 917*
offset, *391*
Oilog, *323*, *502*
oilog, *325, 326*, *507*
oipospoisson, *650*
Oizeta, *243*, *345*, *914*
oizeta, *347, 348*, *916, 917*
oldClass, *809*
olym08, *404*
olym08 *(*olympics*)*, 605
olym12, *404*
olym12 *(*olympics*)*, 605
olympics, 605
Ops.SurvS4 *(*SurvS4*)*, 807
Opt, *153, 154*, *555*, 607, *818*
optim, *144*, *146, 147*, *150*, *152*, *156*, *692, 693*, *695*, *745, 746*
optimize, *209, 210*
options, *152*, *212*, *860*
order, *188*
ordered, *23*, *221*, *223*, *610*, *688*, *792*
ordpoisson, 608, *650*
ordsup, *225*, 610, *842*
Otlog, *323*, *502*
otlog, *325, 326*, *507*
otpospoisson, *650*, *675*
Otzeta, *345*, *914*
otzeta, *347, 348*, *916, 917*
oxtemp, *387*, 612

pairs, *877*
palap *(*alaplaceUC*)*, 32
par, *238, 239*, *414*, *417*, *540–543*, *545, 546*, *562*, *627–629*, *632, 633*, *635*, *637*, *639*, *645*, *689*, *697*, *728*, *788, 789*, *833, 834*

Paralogistic, 613, *615*
paralogistic, *82*, *229*, *304*, *362*, *452*, *455*, *532*, *613, 614*, 614, *762*, *766*
param.names *(*UtilitiesVGAM*)*, 843
Pareto, 616, *619*, *621*
paretoff, *397*, *616, 617*, 617, *623*
ParetoI *(*ParetoIV*)*, 619
ParetoII *(*ParetoIV*)*, 619
paretoII *(*paretoIV*)*, 621
ParetoIII *(*ParetoIV*)*, 619
paretoIII *(*paretoIV*)*, 621
ParetoIV, *617*, 619, *623*
paretoIV, *618–621*, 621
paste, *843*
pbenf *(*Benford*)*, 62
pbenini *(*Benini*)*, 64
pbeta, *944*
pbetabinom *(*Betabinom*)*, 67
pbetabinom.ab, *68*
pbetageom *(*Betageom*)*, 78
pbetanorm *(*Betanorm*)*, 82
pbiamhcop *(*Biamhcop*)*, 87
pbifgmcop *(*Bifgmcop*)*, 94
pbifrankcop *(*Frank*)*, 312
pbilogis *(*bilogis*)*, 101
pbinom, *321*, *918*
pbinorm, *113*, *119, 120*
pbinorm *(*Binorm*)*, 116
pbinormcop *(*Binormcop*)*, 121
pbiplackcop *(*Biplackett*)*, 122
pbisa, *128*
pbisa *(*Bisa*)*, 125
pcard *(*Card*)*, 158
pchisq, *536*
pdagum *(*Dagum*)*, 226
pdiffzeta *(*Diffzeta*)*, 242
peexp *(*Expectiles-Exponential*)*, 261
penorm *(*Expectiles-Normal*)*, 262
Perks, 624
perks, *624*, 625, *762*
persp, *628, 629*
perspqrrvglm, *216*, *542, 543*, 627, *832*
peunif *(*Expectiles-Uniform*)*, 265
pexp, *261*
pexpgeom *(*expgeom*)*, 272
pexplog *(*explog*)*, 277
pexppois *(*exppois*)*, 281
pextbetabinom *(*Extbetabinom*)*, 284

pfisk (Fisk), 301
pfoldnorm (Foldnorm), 308
pfrechet (Frechet), 314
pgaitdbinom (Gaitdbinom), 319
pgaitdlog (Gaitdlog), 322
pgaitdnbinom (Gaitdnbinom), 327
pgaitdpois (Gaitdpois), 333
pgaitdzeta (Gaitdzeta), 343
pgamma, *630–632*
pgamma.deriv, 629, *631, 632*, *838, 839*
pgamma.deriv.unscaled, *630*, 631, *838, 839*
pgengamma.stacy (gengammaUC), 364
pgenpois0 (Genpois0), 366
pgenpois1 (Genpois1), 368
pgenpois2 (Genpois1), 368
pgenray (genray), 375
pgensh (Gensh), 378
pgev (gevUC), 387
pgompertz (Gompertz), 392
pgpd (gpdUC), 398
pgumbel, *181*
pgumbel (gumbelUC), 412
pgumbelII (Gumbel-II), 409
phuber (dhuber), 240
phzeta (Hzeta), 439
pinv.gaussian (Inv.gaussian), 446
pinv.lomax (Inv.lomax), 449
pinv.paralogistic (Inv.paralogistic), 452
pkumar (Kumar), 463
plaplace (laplaceUC), 470
plgamma (lgammaUC), 480
plind (Lindley), 481
plino (Lino), 488
plog, *242*, *323*
plog (Log), 501
ploglap (loglapUC), 517
plomax (Lomax), 529
plot, *238*, *414*, *562*, *627, 628*, *645*, *788, 789*, *833*, *877*
plot.default, *238, 239*, *639*
plot.vgam (plotvgam), 640
plot.window, *639*
plotdeplot.lmscreg, *232, 233*, 632
plotdgaitd, *240*, *333*, *343*, *789*
plotdgaitd (plotdgaitd.vglm), 634
plotdgaitd.vglm, 634
plotqrrvglm, 635
plotqtplot.lmscreg, 636, *698, 699*
plotrcim0, *404*, *579, 580*, 638, *710*
plotvgam, 640, *644, 645*, *720*, *739*, *853*, *855*, *862*, *866*
plotvgam.control, *642*, 643, *645*
plotvglm, *642*, 644, *864*
pmakeham, *548*
pmakeham (Makeham), 548
pmaxwell (Maxwell), 555
pnaka (Nakagami), 587
pnbinom, *328*
pneumo, *24*, *221*, *224, 225*, *583*, 646, *793*
pnorm, *32*, *62*, *64*, *116, 117*, *120*, *126, 127*, *159*, *226*, *241*, *258, 259*, *263*, *282*, *301*, *308*, *310*, *365*, *375*, *392*, *409*, *439*, *450*, *453*, *464*, *470*, *480*, *482*, *489*, *517*, *529*, *548*, *556*, *587*, *613*, *616*, *620*, *624*, *672*, *706*, *724*, *763*, *772*, *819*, *823*, *829*, *837*, *895*
pnorm2 (Binorm), 116
points, *788*
Poisson, *334*, *336*, *366*, *371*, *373*, *375*, *650*, *913*, *933*
poisson, *284*, *650*
poisson.points, *558*, 647, *650, 651*, *709*
poissonff, *44, 45*, *133*, *144, 145*, *149*, *154*, *171, 172*, *210*, *212*, *216*, *281*, *340*, *343*, *368*, *371*, *373*, *375*, *404*, *418–420*, *446*, *547*, *551, 552*, *572*, *597*, *600*, *609, 610*, *648*, 648, *652*, *675*, *695*, *714*, *721*, *762*, *768*, *776*, *789, 790*, *845, 846*, *852*, *863*
PoissonPoints, 650
polf, *610*, *650*
Polono, 651
poly, *552*, *783–785*, *827*
polya, *762*
polya (negbinomial), 592
polyaR, *762*
polyaR (negbinomial), 592
posbernoulli.b, *231*, *524*, 653, *657, 658*, *660–663*, *665*
posbernoulli.t, *58, 59*, *231*, *653, 654*, 656, *657*, *659–665*, *755*
posbernoulli.tb, *26*, *653, 654*, *657, 658*, 659, *663–665*, *811*
posbernUC, 662
Posbinom, *321*

posbinomial, *26*, *115*, *199*, *654*, *657*, *658*, *660*, *661*, 664, *762*, *899*, *900*, *921*
Posgeom, 666
Posnegbin, *329*
posnegbinomial, *551*, *597*, 667, *675*, *762*, *907*, *911*, *928*
Posnorm, 670
posnormal, *671*, 672, *762*, *815*, *842*
Pospois, *336*, *675*
pospoisson, *343*, *650*, *669*, 674, *762*, *910*, *911*
powerlink, *444*, *486*, 675, *720*
pparalogistic (Paralogistic), 613
ppareto (Pareto), 616
pparetoI (ParetoIV), 619
pparetoII (ParetoIV), 619
pparetoIII (ParetoIV), 619
pparetoIV (ParetoIV), 619
pperks (Perks), 624
ppoints, *597*
ppois, *171*, *336*
ppolono (Polono), 651
pposgeom (Posgeom), 666
pposnorm (Posnorm), 670
prats, 676
prayleigh (Rayleigh), 706
predict, *13*, *141*, *142*, *681*
predict.bs, *785*
predict.lm, *758*
predict.poly, *785*
predictqrrvglm, *216*, 678
predictvglm, *305*, *575*, *578*, *678*, 679, *864*
prentice74, *364*, *479*, *481*, 681
price (Rice), 724
prinia, *654*, *658*, *661*, 683
probitlink, *35*, *56*, *107*, *108*, *163*, *180*, *181*, *183*, *221*, *222*, *225*, *308*, *357*, *444*, *486*, *511*, *514*, *602*, *611*, 684, *704*, *705*, *793*
profile, *687*
profile.glm, *205*, *206*, *686*, *687*, *877*
profile.nls, *877*
profilevglm, *205*, *421*, *536*, 686, *877*
propodds, *209*, *224*, *225*, *292*, *458*, *552*, *611*, 687, *689*, *705*
prplot, *225*, 688
psc.t2 (Expectiles-sc.t2), 264
psinmad (Sinmad), 763
pslash, *773*
pslash (Slash), 772
pt, *131*, *264*
ptobit (Tobit), 811
ptopple (Topple), 819
ptriangle (Triangle), 822
ptruncpareto (Truncpareto), 836
punif, *266*, *314*, *388*, *399*, *412*
put.smart, 690, *783*–*785*
pyules (Yules), 894
pzabinom (Zabinom), 897
pzageom (Zageom), 900
pzanegbin (Zanegbin), 903
pzapois (Zapois), 908
pzeta, *345*
pzeta (Zeta), 913
pzibinom (Zibinom), 918
pzigeom (Zigeom), 922
pzinegbin (Zinegbin), 925
pzipf (Zipf), 933
pzipfmb (Zipfmb), 936
pzipois (Zipois), 937
pzoabeta (Zoabeta), 943
pzoibetabinom (Betabinom), 67

qalap (alaplaceUC), 32
qbenf (Benford), 62
qbenini (Benini), 64
qbetanorm (Betanorm), 82
qbinom, *321*
qbisa (Bisa), 125
qcard (Card), 158
qdagum (Dagum), 226
qdiffzeta (Diffzeta), 242
qeexp (Expectiles-Exponential), 261
qenorm (Expectiles-Normal), 262
qeunif (Expectiles-Uniform), 265
qexp, *261*
qexpgeom (expgeom), 272
qexplog (explog), 277
qexppois (exppois), 281
qextbetabinom (Extbetabinom), 284
qfisk (Fisk), 301
qfoldnorm (Foldnorm), 308
qfrechet (Frechet), 314
qgaitdbinom (Gaitdbinom), 319
qgaitdlog (Gaitdlog), 322
qgaitdnbinom (Gaitdnbinom), 327
qgaitdpois (Gaitdpois), 333
qgaitdzeta (Gaitdzeta), 343

qgamma, *349*
qgengamma.stacy *(*gengammaUC*)*, 364
qgenpois0 *(*Genpois0*)*, 366
qgenpois1 *(*Genpois1*)*, 368
qgenpois2 *(*Genpois1*)*, 368
qgenray *(*genray*)*, 375
qgensh *(*Gensh*)*, 378
qgev *(*gevUC*)*, 387
qgompertz *(*Gompertz*)*, 392
qgpd *(*gpdUC*)*, 398
qgumbel *(*gumbelUC*)*, 412
qgumbelII *(*Gumbel-II*)*, 409
qhuber *(*dhuber*)*, 240
qhzeta *(*Hzeta*)*, 439
qinv.lomax *(*Inv.lomax*)*, 449
qinv.paralogistic *(*Inv.paralogistic*)*, 452
qkumar *(*Kumar*)*, 463
qlaplace *(*laplaceUC*)*, 470
qlgamma *(*lgammaUC*)*, 480
qlino *(*Lino*)*, 488
qlog, *323*
qlog *(*Log*)*, 501
qlogis, *183*
qloglap *(*loglapUC*)*, 517
qlomax *(*Lomax*)*, 529
qmakeham *(*Makeham*)*, 548
qmaxwell *(*Maxwell*)*, 555
qnaka *(*Nakagami*)*, 587
qnbinom, *328*, *593*
qnorm, *32*, *62*, *64*, *126*, *159*, *183*, *226*, *241*, *263*, *282*, *301*, *308*, *365*, *375*, *392*, *409*, *439*, *450*, *453*, *464*, *470*, *480*, *482*, *489*, *517*, *529*, *548*, *556*, *587*, *613*, *616*, *620*, *624*, *706*, *721*, *724*, *763*, *772*, *819*, *823*, *837*, *895*
qparalogistic *(*Paralogistic*)*, 613
qpareto *(*Pareto*)*, 616
qparetoI *(*ParetoIV*)*, 619
qparetoII *(*ParetoIV*)*, 619
qparetoIII *(*ParetoIV*)*, 619
qparetoIV, *622*
qparetoIV *(*ParetoIV*)*, 619
qperks *(*Perks*)*, 624
qpois, *336*
qposgeom *(*Posgeom*)*, 666
qposnorm *(*Posnorm*)*, 670
qrayleigh *(*Rayleigh*)*, 706
qrice *(*Rice*)*, 724
qrrvglm.control, *156*, *157*, *212–216*, *350*, *576*, *593*, 691, *713*, *714*
qsc.t2 *(*Expectiles-sc.t2*)*, 264
qsinmad *(*Sinmad*)*, 763
qt, *264*
qtobit *(*Tobit*)*, 811
qtopple *(*Topple*)*, 819
qtplot.gumbel, 696
qtplot.gumbelff *(*qtplot.gumbel*)*, 696
qtplot.lmscreg, *167*, *233*, *494*, *497*, *500*, *638*, 698
qtriangle *(*Triangle*)*, 822
qtruncpareto *(*Truncpareto*)*, 836
quantile, *196*, *198*, *567*, *569*, *571*, *595*, *774*, *777*, *778*
quasipoisson, *371*, *373*, *375*, *594*, *597*
qunif, *266*, *314*, *388*, *399*, *412*
Qvar, *277*, *404*, 699, *703*, *842*
qvar, *701*, 703
qyules *(*Yules*)*, 894
qzabinom *(*Zabinom*)*, 897
qzageom *(*Zageom*)*, 900
qzanegbin *(*Zanegbin*)*, 903
qzapois *(*Zapois*)*, 908
qzeta, *345*
qzeta *(*Zeta*)*, 913
qzibinom *(*Zibinom*)*, 918
qzigeom *(*Zigeom*)*, 922
qzinegbin *(*Zinegbin*)*, 925
qzipf *(*Zipf*)*, 933
qzipfmb *(*Zipfmb*)*, 936
qzipois *(*Zipois*)*, 937
qzoabeta *(*Zoabeta*)*, 943

R2latvar, *225*, *688*, 704
ralap, *30*
ralap *(*alaplaceUC*)*, 32
range, *835*, *836*
Rank, 705
rank, *705*
Rayleigh, *556*, 706, *709*
rayleigh, *364*, *376*, *377*, *556*, *558*, *589*, *647*, *648*, *707*, 707, *709*, *726*, *762*, *886*, *889*
rbell, *62*
rbenf *(*Benford*)*, 62
rbenini *(*Benini*)*, 64
rbetabinom *(*Betabinom*)*, 67

rbetageom, *77*, *80*, *87*, *382*
rbetageom (Betageom), 78
rbetanorm, *77*, *87*
rbetanorm (Betanorm), 82
rbiamhcop, *89*
rbiamhcop (Biamhcop), 87
rbiclaytoncop, *92*
rbiclaytoncop (Biclaytoncop), 90
rbifgmcop, *96*
rbifgmcop (Bifgmcop), 94
rbifrankcop, *98, 99*
rbifrankcop (Frank), 312
rbilogis, *89*, *104*
rbilogis (bilogis), 101
rbinom, *321*
rbinom2.or, *107, 108*
rbinom2.or (Binom2.or), 104
rbinom2.rho, *113*
rbinom2.rho (Binom2.rho), 109
rbinorm, *829*
rbinorm (Binorm), 116
rbinormcop, *120*
rbinormcop (Binormcop), 121
rbiplackcop, *124, 125*
rbiplackcop (Biplackett), 122
rbisa (Bisa), 125
rbort, *133, 134*
rbort (Bort), 134
rcard, *161*
rcard (Card), 158
Rcim, *404*, *579, 580*, *640*, 709
rcim, *13*, *16*, *61*, *219*, *276, 277*, *579, 580*, *639, 640*, *699–701*, *703*, *710*, *818*, *872*, *874*
rcim (grc), 401
rcqo, *216*, *695*, 711
rdagum (Dagum), 226
rdiffzeta (Diffzeta), 242
rdiric, *245*, 715
rec.exp1, 716
rec.normal, *420*, 717
reciprocal, *357*
reciprocallink, *486*, *593*, *595*, 719
reexp (Expectiles-Exponential), 261
renorm (Expectiles-Normal), 262
resid, *722*
residualsvglm, *115*, *342, 343*, *650*, 720, *864*
reunif (Expectiles-Uniform), 265
rexp, *261*, *280*, *568*
rexpgeom (expgeom), 272
rexplog (explog), 277
rexppois (exppois), 281
rextbetabinom (Extbetabinom), 284
rfisk (Fisk), 301
rfoldnorm, *311*
rfoldnorm (Foldnorm), 308
rfrechet, *316*
rfrechet (Frechet), 314
rgaitdbinom (Gaitdbinom), 319
rgaitdlog (Gaitdlog), 322
rgaitdnbinom (Gaitdnbinom), 327
rgaitdpois (Gaitdpois), 333
rgaitdzeta (Gaitdzeta), 343
rgamma, *260*, *349*, *351*, *356*
rgengamma.stacy, *364*
rgengamma.stacy (gengammaUC), 364
rgenpois0 (Genpois0), 366
rgenpois1 (Genpois1), 368
rgenpois2 (Genpois1), 368
rgenray (genray), 375
rgensh (Gensh), 378
rgeom, *666*, *925*
rgev, *387*
rgev (gevUC), 387
rgompertz (Gompertz), 392
rgpd, *397*
rgpd (gpdUC), 398
rgumbel, *169*, *408*
rgumbel (gumbelUC), 412
rgumbelII (Gumbel-II), 409
rhobitlink, *112, 113*, *118*, *286*, *300, 301*, *371*, *486*, *559*, *602*, 722
rhuber, *428*
rhuber (dhuber), 240
rhzeta (Hzeta), 439
Rice, 724
riceff, *709*, *724*, 725, *762*
rig, *761*
rigff, 727
rinv.gaussian, *885*
rinv.gaussian (Inv.gaussian), 446
rinv.lomax (Inv.lomax), 449
rinv.paralogistic (Inv.paralogistic), 452
rkumar (Kumar), 463
rlaplace, *470*

rlaplace (laplaceUC), 470
rlgamma, *479*
rlgamma (lgammaUC), 480
rlind (Lindley), 481
rlino (Lino), 488
rlog, *323*
rlog (Log), 501
rlogis, *101*, *509*
rloglap (loglapUC), 517
rlomax (Lomax), 529
rlplot.gev (rlplot.gevff), 728
rlplot.gevff, *387*, 728
rmakeham (Makeham), 548
rmaxwell (Maxwell), 555
rnaka, *589*
rnaka (Nakagami), 587
rnbinom, *328*, *593*, *595*, *596*, *599*, *600*, *669*, *926*
RNG, *762*
rnorm, *90*, *116*, *121*, *263*, *308*, *671*, *812*, *815*, *828*
rootogram4, *326*, *333*, *343*, *348*, 730
rootogram4vglm (rootogram4), 730
Round, *28*, *37*, *40*, *42*, *44*
round, *732*
round2, 732, *814*, *815*
rownames, *403*
rowSums, *339*
rparalogistic (Paralogistic), 613
rpareto (Pareto), 616
rparetoI (ParetoIV), 619
rparetoII (ParetoIV), 619
rparetoIII (ParetoIV), 619
rparetoIV (ParetoIV), 619
rperks (Perks), 624
rpois, *280*, *336*, *572*, *929*, *942*
rpois.points (PoissonPoints), 650
rpolono (Polono), 651
rposbern, *654*, *658*
rposbern (posbernUC), 662
rposgeom, *901*
rposgeom (Posgeom), 666
rposnorm (Posnorm), 670
rrar, 733
rrayleigh (Rayleigh), 706
rrice (Rice), 724
rrvglm, *13*, *16*, *23*, *61*, *84*, *86*, *103*, *114*, *115*, *149–151*, *153*, *161*, *178*, *179*, *183*, *189–191*, *216*, *219*, *221*, *223*, *224*, *245*, *247*, *249*, *326*, *332*, *341*, *347*, *401*, *403*, *404*, *424*, *436*, *448*, *473*, *487*, *493*, *496*, *508*, *521*, *522*, *537*, *546*, *557*, *559*, *583–585*, *596*, *597*, *599*, *641*, *647*, *649*, *650*, *668*, *674*, *708*, *734*, 735, *739*, *741*, *743*, *744*, *749*, *785*, *792*, *797*, *799*, *810*, *864*, *874*, *876*, *941*, *942*
rrvglm-class, 738
rrvglm.control, *215*, *402–404*, *546*, *695*, *735–737*, 741, *746*, *799*
rrvglm.optim.control, *743*, *744*, 745
rsc.t2 (Expectiles-sc.t2), 264
rsimplex (Simplex), 759
rsinmad, *199*
rsinmad (Sinmad), 763
rskellam (Skellam), 766
rskewnorm (skewnorm), 769
rslash, *775*
rslash (Slash), 772
rstandard, *721*
rtobit, *815*
rtobit (Tobit), 811
rtopple (Topple), 819
rtriangle (Triangle), 822
rtrinorm, *830*
rtrinorm (Trinorm), 828
rtruncpareto (Truncpareto), 836
ruge, *650*, 746
runif, *64*, *67*, *68*, *78*, *83*, *87*, *94*, *104*, *109*, *122*, *126*, *159*, *242*, *264*, *266*, *284*, *285*, *313*, *314*, *364*, *366*, *368*, *392*, *409*, *439*, *470*, *480*, *482*, *489*, *501*, *548*, *587*, *616*, *620*, *624*, *662*, *666*, *706*, *721*, *724*, *759*, *767*, *769*, *772*, *823*, *837*, *918*, *922*, *926*
rweibull, *198*
ryules, *896*
ryules (Yules), 894
rzabinom (Zabinom), 897
rzageom (Zageom), 900
rzanegbin (Zanegbin), 903
rzapois, *911*
rzapois (Zapois), 908
rzeta, *345*
rzeta (Zeta), 913
rzibinom, *920*, *921*

rzibinom (Zibinom), 918
rzigeom, *925*
rzigeom (Zigeom), 922
rzinegbin, *938*
rzinegbin (Zinegbin), 925
rzipf (Zipf), 933
rzipfmb (Zipfmb), 936
rzipois, *926*
rzipois (Zipois), 937
rzoabeta (Zoabeta), 943
rzoibetabinom (Betabinom), 67

s, *14*, *193*, *297*, *386*, *396, 397*, *456*, *747*, 747, *777–779*, *781*, *801*, *850–853*, *856*, *858*, *880*
sc.studentt2, *30*, *264, 265*, 749, *796*
scale, *153*, *639*, *695*, *784, 785*
score.stat, *421*, *536*, 750, *804*, *884*
score.stat.vlm, *48*, *420*, *537*, *803*, *864*
seglines, *423*, 752
Select, *297*, *404*, *658*, *661*, 753, *869, 870*
seq2binomial, *115*, *174*, 756
set.seed, *154*, *214*, *216*, *713*, *721*
setdiff, *836*
setMethod, *552*, *804*
setup.smart, 757, *785*, *893*
ships, *701*, *703*
show,SurvS4-method (SurvS4-class), 809
show.summary.drrvglm (summary.drrvglm), 797
show.summary.pvgam (summarypvgam), 800
show.summary.rrvglm (summary.drrvglm), 797
show.summary.vgam (summaryvgam), 801
show.summary.vglm (summaryvglm), 802
show.SurvS4 (SurvS4), 807
Simplex, 759
simplex, *115*, *245*, *727*, *759*, 760, *762*
simulate, *761*, *873*
simulate.vlm, *30*, *72*, *75*, *77*, *87*, *89*, *96*, *99*, *115*, *165*, *229*, *260*, *281*, *304*, *326*, *333*, *343*, *348, 349*, *351*, *356*, *364*, *382*, *394*, *439*, *441*, *452*, *455*, *466*, *484*, *507*, *509*, *527*, *532*, *550*, *597*, *600*, *626*, *650*, *665*, *669*, *675*, *709*, *726*, 761, *766*, *775*, *796*, *825*, *842*, *896*, *903*, *907*, *911*, *925*, *935*, *942*
Sinmad, 763, *766*
sinmad, *82*, *199*, *229*, *304*, *361, 362*, *452*, *455*, *532*, *615*, *762–764*, 764
Skellam, 766
skellam, *650*, *767*, 767
skewnorm, 769, *771*
skewnormal, *311*, *769, 770*, 770, *842*
Slash, 772
slash, *762*, *772, 773*, 773
sloglink, *35*, *56*, 775, *790*
sm.bs, *383*, *827*
sm.bs (smartpred), 784
sm.ns, *827*
sm.ns (smartpred), 784
sm.os, *13, 14*, *748*, 776, *778*, *780, 781*, *800, 801*, *850–853*, *857, 858*, *880*
sm.poly, *383*, *827*
sm.poly (smartpred), 784
sm.ps, *13, 14*, *456*, *748*, *778, 779*, 780, *785*, *800, 801*, *850–853*, *857, 858*
sm.scale (smartpred), 784
smart.expression, 783, *785*
smart.mode.is, 783, *785*
smartpred, *153*, *213*, *575*, *578*, *680, 681*, *736, 737*, *779*, *781*, 784, *852*, *861*, *863, 864*
smooth.spline, *777*, *779*, *879, 880*
specials, *333*, *336*, *343*, 786
specialsvglm, *36, 37*
specialsvglm (specials), 786
spikeplot, *240*, *326*, *333*, *336*, *340, 341*, *343*, *348*, *634*, 787, *900*, *903*, *907*, *911*, *921*, *925*, *929*, *942*
splineDesign, *778, 779*, *781*
sqrt, *789*
sqrtlink, *55*, *486*, *775, 776*, 789
sratio, *24*, *183*, *197*, *199*, *221*, *223–225*, *458*, *551, 552*, *585*, *721*, 791, *792*, *873*
stat.anova, *48*
stats:dt, *378*
step, *794*
step4, 794
step4vglm, *25*, *578*, *827*, *864*
step4vglm (step4), 794
stop, *48*, *240*
structure, *809*
studentt, *165*, *508*, *762*, 795, *842*
studentt2, *749, 750*, *762*
studentt2 (studentt), 795

studentt3, *762*
studentt3 *(*studentt*)*, 795
subset, *754, 755*, *850*
subsetcol *(*Select*)*, 753
sum, *519*
summary, *153*, *804*
summary.drrvglm, *737*, *744*, *797*, 797
summary.gam, *800–802*
summary.glm, *751*, *801–804*, *883, 884*
summary.lm, *705*, *801, 802*, *804*
summary.rrvglm, *797*, *804*
summary.rrvglm *(*summary.drrvglm*)*, 797
Summary.SurvS4 *(*SurvS4*)*, 807
summarypvgam, *779*, *781*, 800, *802*, *853*
summaryvgam, *801*, 801, *853*
summaryvglm, *14*, *206*, *420, 421*, *535, 536*, *751*, *797–801*, 802, *826, 827*, *848*, *864*, *883, 884*
SURff, *390*, 805, *842*
Surv, *171*
survreg, *808, 809*
SurvS4, *171, 172*, 807, *809*, *888*
SurvS4-class, 809

table, *788, 789*
TDist, *796*
term.names *(*formulavlm*)*, 311
term.namesvlm *(*formulavlm*)*, 311
terms, *740*, *855*, *863*, *866*
TIC, 810
TICvlm, *27*
TICvlm *(*TIC*)*, 810
title, *238*, *627–629*, *697*, *728*, *833, 834*
Tobit, 811
tobit, *170*, *252*, *566*, *673*, *732*, *812*, 813, *842*
Tol, *555*, *608*, 817
Topple, 819, *821*, *825*
topple, *418*, *819, 820*, 820, *823*
toxop, *254*, 821
trapO, *154*, *216*, *467*
Triangle, *820, 821*, 822, *825*
triangle, *762*, *823*, 824
trigamma, *594*
trim.constraints, *25*, *208*, *578*, *794*, 826
Trinorm, 828
trinormal, *119*, *829*, 829, *842*
trplot, *539*, 831
trplot.qrrvglm, *216*, *832*, 832
Trunc, *326*, *333*, *343*, *348*, *391*, 835
truncated, *787*
truncated *(*altered*)*, 36
truncgeometric *(*geometric*)*, 381
Truncpareto, *619*, 836
truncpareto, *837*
truncpareto *(*paretoff*)*, 617
truncweibull, *632*, 838, *886*, *889*
TypicalVGAMfamilyFunction, *16*, *487*, *744*, *864*, *870*
TypicalVGAMfamilyFunction *(*CommonVGAMffArguments*)*, 194
TypicalVGAMlink *(*Links*)*, 485

ucberk, 840
Uniform, *433*, *556*, *819*
uninormal, *49*, *117*, *119*, *170*, *178*, *252*, *276, 277*, *286*, *310, 311*, *372*, *403*, *418*, *424*, *428*, *527*, *534*, *570*, *601, 602*, *611*, *673*, *685*, *699*, *701*, *703*, *718*, *762*, *771*, *796*, *806*, *814, 815*, *829, 830*, 841, *842*
uniroot, *42*, *144*, *146*, *150*, *308, 309*, *587*
update, *25*, *48*, *537*, *794*, *864*
uqo, *213*, *216*
uqo *(*grc*)*, 401
UtilitiesVGAM, *201*, 843

V1, *650*, 844, *846*
V2, *650*, *845*, 845
valt0.control, *743*
vcov, *799*, *847, 848*
vcovdrrvglm, *799*
vcovdrrvglm *(*vcovvlm*)*, 846
vcovqrrvglm, *216*, *576*, *847*
vcovqrrvglm *(*vcovvlm*)*, 846
vcovrrvglm, *847*
vcovrrvglm *(*vcovvlm*)*, 846
vcovvlm, *206*, *804*, 846, *847*
vector, *809*
venice, *169*, *387*, *408*, *415*, 848
venice90 *(*venice*)*, 848
VGAM *(*VGAM-package*)*, 13
vgam, *13*, *16*, *18–21*, *23*, *29*, *38*, *40*, *42*, *45*, *50*, *66*, *77*, *80*, *82*, *84*, *86*, *89*, *91*, *96–98*, *100*, *103*, *107*, *112*, *114*, *118*, *120*, *124*, *128*, *130*, *133*, *153*, *161*, *165, 166*, *168*, *170, 171*, *178*, *192, 193*, *200, 201*, *208*, *221*, *223, 224*, *229*, *232*, *243*, *245*, *247*, *249*, *252*, *260*,

269, 270, 274, 279, 280, 283, 290, 292, 294, 295, 297, 303, 310, 316, 318, 326, 332, 341, 347, 349, 351, 353, 354, 356, 361, 363, 370, 373, 374, 377, 380, 381, 385, 386, 394, 396, 397, 407, 411, 428, 435, 436, 438, 440, 445, 448, 451, 454–456, 465, 469, 472, 473, 477, 479, 483, 487, 491, 493, 496, 499, 505, 506, 508, 515, 521, 522, 527, 531, 534, 537, 539, 550, 552, 557, 559, 565, 567, 570, 572, 574, 584, 589, 596, 599, 602, 605, 609, 615, 618, 623, 626, 637, 641, 642, 647, 649, 654, 657, 660, 664, 668, 673, 674, 682, 687, 697, 698, 708, 717, 718, 726, 727, 731, 733, 736, 747–749, 756, 757, 760, 762, 765, 768, 771, 774, 778, 779, 781, 785, 792, 796, 800–802, 805, 810, 814, 821, 824, 830, 839, 841, 850, 857–859, 864, 872, 874, 876, 879, 885, 886, 888, 890, 896, 899, 902, 906, 910, 917, 921, 924, 928, 932, 935, 941

vgam-class, 854

VGAM-package, 13

vgam.control, 155, 850–853, 856, 857

vglm, 13, 16, 18–21, 23–25, 29, 37, 38, 40, 42, 45, 46, 48, 50, 52, 66, 71, 74, 77, 80, 82, 84, 86, 89, 91, 93, 96–98, 100, 103, 107, 112, 114, 118, 120, 124, 128, 130, 133, 137, 139, 161, 165, 166, 168, 170, 171, 178, 182, 185, 193, 194, 197, 198, 200, 201, 205, 208, 209, 215, 216, 221, 223, 224, 229, 232, 234, 236, 243, 245, 247, 249, 252, 254, 257, 260, 269, 270, 274, 279, 280, 283, 287, 290, 292, 294–297, 303, 306, 310, 316, 318, 326, 332, 341, 347, 349, 351, 353, 354, 356–358, 361, 363, 370, 373, 374, 377, 380, 381, 385, 386, 391, 394, 396, 397, 401, 403, 407, 411, 416–418, 421, 428, 435, 436, 438, 440, 445, 448, 451, 454, 456–458, 460, 465, 469, 472, 473, 477, 479, 483, 485, 487, 491, 493, 496, 499, 505, 506, 508, 515, 521, 522, 527, 531, 534–537, 539, 550–552, 557, 559, 565, 567, 570, 572, 574, 578, 584, 589, 596, 597, 599, 602, 609, 611, 615, 618, 623, 626, 637, 641, 642, 645, 647, 649, 654, 657, 660, 664, 668, 673, 674, 679–682, 687, 697–701, 704–706, 708, 717, 718, 720, 722, 726–728, 731, 733–737, 744, 747–749, 751, 755–757, 760, 762, 765, 768, 771, 774, 779, 785, 787, 792, 794, 796, 802, 804–806, 808–810, 814, 821, 824, 826, 827, 830, 839, 841, 847, 850, 851, 853, 856, 858, 859, 859, 867, 870, 872, 874, 876, 882, 884–886, 888, 890, 891, 896, 899, 902, 906, 910, 917, 921, 924, 928, 932, 935, 941

vglm-class, 865

vglm.control, 50, 108, 205, 224, 291, 297, 306, 361, 362, 388, 403, 584, 642, 686, 695, 739, 743, 744, 805, 852, 855, 857–861, 863, 864, 866, 868, 883, 929, 932

vglmff-class, 872

vonmises, 161, 875

vpairs.profile (vplot.profile), 877

vplot.profile, 877

vsmooth.spline, 28, 495, 748, 853, 859, 878

waitakere, 432, 880

wald.stat, 206, 421, 536, 751, 804, 882

wald.stat.vlm, 48, 420, 535–537, 751, 803, 864

waldff, 447, 449, 884

weibull.mean, 885, 889

weibullR, 198, 269, 364, 387, 410, 411, 709, 838, 839, 886, 887

weightsvglm, 150, 862, 890

wine, 891

wrapup.smart, 785, 892

xs.nz, 583

yeo.johnson, 893

yip88, 910, 941, 942

Yules, 894

yulesimon, 762, 895, 896

Zabinom, 897

zabinomial, 898, *921*
zabinomialff *(*zabinomial*)*, 898
Zageom, 900
zageometric, *382*, *666*, *762*, *901*, 901, *924, 925*
zageometricff, *762*
zageometricff *(*zageometric*)*, 901
Zanegbin, 903
zanegbinomial, *669*, *762*, *904*, 905, *929*
zanegbinomialff, *762*
zanegbinomialff *(*zanegbinomial*)*, 905
Zapois, *336*, 908
zapoisson, *340*, *343*, *675*, *731*, *762*, *909*, 909, *941, 942*
zapoissonff, *340*, *762*
zapoissonff *(*zapoisson*)*, 909
zero, *208*, 911
Zeta, 913, *916, 917*
zeta, *244*, *439*, *441*, *476*, *914*, 914, *917*
zetaff, *243, 244*, *348*, *439*, *441*, *913–916*, 916, *933–935*
zetaffMlink, *347, 348*, *391*
Zibinom, 918
zibinomial, *115*, *898–900*, *918, 919*, 919
zibinomialff *(*zibinomial*)*, 919
Zigeom, 922
zigeometric, *382*, *666*, *762*, *901*, *903*, *923*, 923
zigeometricff, *762*
zigeometricff *(*zigeometric*)*, 923
Zinegbin, 925, *929*
zinegbinomial, *196*, *597*, *737*, *762*, *907*, *926*, 927, *941*
zinegbinomialff *(*zinegbinomial*)*, 927
ziP, *341*
zipebcom, *108*, 930, *942*
Zipf, 933, *937*
zipf, *243, 244*, *762*, *917*, *933, 934*, 934
Zipfmb, *934*, 936
Zipois, *335, 336*, *919*, *923*, *926*, 937, *941, 942*
zipoisson, *36*, *340*, *343*, *401*, *403*, *418*, *650*, *674, 675*, *731*, *737*, *762*, *907*, *910, 911*, *921*, *924*, *929–932*, *938*, 939, *946*
zipoissonff, *340*, *401*, *403*, *418*, *532*, *762*
zipoissonff *(*zipoisson*)*, 939
Zoabeta, *69*, 943, *946*
zoabetaR, *944*, 945