Package 'BurStFin'

October 12, 2022

Version 1.3

Date 2022-04-16

Title Burns Statistics Financial

Author Burns Statistics

Maintainer Pat Burns <patrick@burns-stat.com>

Depends stats

Description A suite of functions for finance, including the estimation of variance matrices via a statistical factor model or Ledoit-Wolf shrinkage.

License Unlimited

URL https://www.burns-stat.com/

NeedsCompilation no

Repository CRAN

Date/Publication 2022-04-18 12:12:29 UTC

R topics documented:

`alpha.proxy` *Compute and Plot Alpha Proxy*

Description

Computes (and possibly generates a contour plot of) the alpha proxy – a measure of the effect that volatility and correlation have on the utility of the investor. As the name might suggest, it is in the same units as alpha (that is, expected returns).

Usage

```
alpha.proxy(weight = 0.2, vol.man = 0.2, vol.bench = 0.2,
vol.other = 0.2, cor.man = 0.2, cor.bench = 0.2,
plot.it = TRUE, transpose = FALSE, ...)
```

Arguments

`weight`	a number or vector of the fraction of the value of the investor's entire portfolio that is given to the manager.
`vol.man`	a number or vector giving the volatility of the manager's portfolio.
`vol.bench`	a number or vector giving the volatility of the benchmark.
`vol.other`	a number or vector giving the volatility of the rest of the portfolio.
`cor.man`	a number or vector giving the correlation between the manager's portfolio and the rest of the investor's portfolio.
`cor.bench`	a number or vector giving the correlation between the benchmark and the rest of the investor's portfolio.
`plot.it`	logical value. If `TRUE` and two of the arguments above have length greater than 1, then a contour plot is created.
`transpose`	logical value. If `TRUE`, then the variables on the axes of the contour plot are switched.
`...`	additional arguments to `filled.contour` may be given.

Value

a vector of the alpha proxies (in basis points) if less than 2 of the first 6 arguments have length more than 1.

Otherwise it is a list (returned invisibly if `plot.it` is `TRUE`) with the following components:

`x`	one of the vectors of inputs.
`y`	the other vector of inputs.
`z`	a matrix of the computed alpha proxies where rows correspond to the values in `x` and the columns correspond to `y`. The values are given in basis points.
`call`	a character string of the image of the command that created the object.

Side Effects

if `plot.it` is `TRUE`, then a contour plot is created.

An error occurs if more than 2 of the first 6 arguments have length greater than 1.

Details

The first 6 arguments are the variables that determine the alpha proxy. The investor is faced with the decision of hiring the manager for some portion (the weight) of the portfolio rather than using a replication of the benchmark. If the alpha proxy is positive, then the volatility and correlation of the manager's portfolio is improving whatever outperformance the manager may have (or is offsetting the losses).

Revision

This help was last revised 2010 January 05.

References

Burns, Patrick (2003). "Portfolio Sharpening". Working Paper, Burns Statistics http://www.burns-stat.com/.

See Also

`partial.rainbow`.

Examples

```
# return vector of alpha proxies
alpha.proxy(weigh=.05, vol.man=.17, cor.man=seq(0, .2, len=21))

# create a contour plot
alpha.proxy(weigh=.05, vol.man=seq(.15, .25, len=20),
cor.man=seq(0, .2, len=21))

# commands used to create figures in the paper
alpha.proxy(vol.man=seq(.15, .25, len=50), weight=seq(.01, .7, leng=60),
        color.palette=partial.rainbow(start=0, end=.32))

alpha.proxy(cor.man=seq(0, .3, len=50), weight=seq(.01, .7, leng=60),
        color.palette=partial.rainbow(start=.07))

alpha.proxy(cor.man=seq(0, .3, len=50), vol.man=seq(.15, .25, leng=60),
        color.palette=partial.rainbow(start=0))
```

`factor.model.stat` *Estimate Variance Matrix via Statistical Factors*

Description

Creates a variance matrix based on the principal components of the variables that have no missing values.

Usage

```
factor.model.stat(x, weights = seq(0.5, 1.5, length.out = nobs),
output = "full", center = TRUE, frac.var = 0.5, iter.max = 1,
nfac.miss = 1, full.min = 20, reg.min = 40, sd.min = 20,
quan.sd = 0.9, tol = 0.001, zero.load = FALSE,
range.factors = c(0, Inf), constant.returns.okay = FALSE,
specific.floor = 0.1, floor.type = "quantile", verbose=2)
```

Arguments

`x`	required. A numeric matrix. The rows are observations and the columns are the variables. In finance, this will be a matrix of returns where the rows are times and the columns are assets. For the default value of `weights` the most recent observation should be the last row. The number of columns may exceed the number of rows, and missing values are accepted. A column may even have all missing values.
`weights`	a vector of observation weights, or `NULL`. Equal weights can be specified with `NULL` or with a single positive number. Otherwise, the length must be equal to either the original number of rows in `x` or the number of rows in `x` minus the number of rows that contain all missing values.
`output`	a character string indicating the form of the result. It must partially match one of: `"full"`, `"systematic"`, `"specific"` or `"factor"`.
`center`	either a logical value or a numeric vector with length equal to the number of columns in `x`. If `center` is `TRUE`, then the mean of each column is used as the center. If `center` is `FALSE`, then the center for each variable is taken to be zero.
`frac.var`	a control on the number of factors to use – the number of factors is chosen so that the factors account for (just over) `frac.var` of the total variability.
`iter.max`	the maximum number of times to iterate the search for principal factors of the variables with complete data.
`nfac.miss`	a vector of integers giving the number of factors to use in regressions for variables with missing values. The number of factors used is equal to the i-th element of `nfac.miss` where i is the number of missing values for the variable. Thus the values in the vector should be non-increasing. The last value is used when the number of missing values is greater than the length of `nfac.miss`.

`full.min` an integer giving the minimum number of variables that must have complete data.

`reg.min` the minimum number of non-missing values for a variable in order for a regression to be performed on the variable.

`sd.min` the minimum number of non-missing values for a variable in order for the standard deviation to be estimated from the data.

`quan.sd` the quantile of the standard deviations to use for the standard deviation of variables that do not have enough data for the standard deviation to be estimated.

`tol` a number giving the tolerance for the principal factor convergence (using the assets with full data). If the maximum change in uniquenesses (in the correlation scale) is less than `tol` from one iteration to the next, then convergence is assumed and the iterations end.

`zero.load` a logical value. If `TRUE`, then loadings for variables with missing values are zero except for those estimated by regression. If `FALSE`, then loadings for variables with missing values are the average loading for the factor (when they are not estimated by regression).

`range.factors` a numeric vector that gives the maximum and minimum number of factors that are allowed to be used.

`constant.returns.okay`
a logical vector: if `TRUE`, then a column with all of its non-missing values equal does not cause an error.

if the true variance is thought to be non-zero, then a better alternative to setting this to `TRUE` is to set all the values in the column of `x` to be `NA`.

`specific.floor` a number indicating how much uniquenesses should be adjusted upwards. The meaning of this number depends on the value of the `floor.type` argument.

`floor.type` a character string that partially matches one of: `"quantile"` or `"fraction"`.

If the value is `"quantile"`, then all uniquenesses are made to be at least as big as the `specific.floor` quantile of the uniquenesses.

If the value is `"fraction"`, then all uniqueness are made to be at least `specific.floor`.

`verbose` a number indicating the level of warning messages desired. This currently controls warnings:

If at least 1, then a warning will be issued if all the values in `x` are non-negative. In finance this is an indication that prices rather than returns are input (an easy mistake to make).

If at least 1, then a warning will be issued if there are any assets with constant returns (unless `constant.returns.okay` is `FALSE` in which case an error is thrown).

If at least 2, then a warning will be issued if there are any specific variances that are adjusted from being negative.

Value

if `output` is `"full"`, then a variance matrix with dimensions equal to the number of columns in the input `x`. This has two additional attributes: `number.of.factors` that says how many factors are used in the model, and `timestamp` that gives the date and time that the object was created.

if output is "systematic", then a matrix with dimensions equal to the number of columns in the input x that contains the systematic portion of the variance matrix.

if output is "specific", then a diagonal matrix with dimensions equal to the number of columns in the input x that contains the specific variance portion of the variance matrix. The full variance matrix is the sum of the systematic and specific matrices.

If output is "factor", then an object of class "statfacmodBurSt" which is a list with components:

loadings — a matrix of the loadings for the correlation matrix.

uniquenesses — the uniquenesses for the correlation matrix. That is, the proportion of the variance that is not explained by the factors.

Note that if there are uniquenesses that have been modified via the specific.floor argument, then the actual proportion is the stated proportion divided by one plus the modification.

sdev — the standard deviations for the variables.

Note that if there are uniquenesses that have been modified via the specific.floor argument, then the corresponding standard deviations in sdev are smaller than the actual standard deviations in the answer.

constant.names

A character vector giving the names of the variables that are constant (if any).

cumulative.variance.fraction

numeric vector giving the cumulative fraction of the variance explained by (all) the factors.

timestamp — character string giving the date and time the calculation was completed.

call — an image of the call that created the object.

Details

Observations that are missing on all variables are deleted. Then a principal components factor model is estimated with the variables that have complete data.

For variables that have missing values, the standard deviation is estimated when there are enough obeservations otherwise a given quantile of the standard deviations of the other assets is used as the estimate. The loadings for these variables are set to be either the average loading for the variables with no missing data, or zero. The loadings for the most important factors are modified by performing a regression with the non-missing data for each variable (if there is enough data to do the regression).

The treatment of variables with missing values can be quite important. You may well benefit from specializing how missing values are handled to your particular problem. To do this, set the output to "factor" – then you can modify the loadings (and per force the uniquenesses), and the standard deviations to fit your situation. This may include taking sectors and countries into account, for example.

The default settings for missing value treatment are suitable for creating a variance matrix for long-only portfolio optimization – high volatility and average correlation. Take note that the proper treatment of missing values is HIGHLY dependent on the use to which the variance matrix is to be put.

OBSERVATION WEIGHTS. Time weights are quite helpful for estimating variances from returns. The default weighting seems to perform reasonably well over a range of situations.

FACTOR MODEL TO FULL MODEL. This class of object has a method for `fitted` which returns the variance matrix corresponding to the factor model representation.

Warning

The default value for `weights` assumes that the last row is the most recent observation and the first observation is the most ancient observation.

Research Issues

The method of handling missing values used in the function has not been well studied. It seems not to be the worst approach, but undoubtedly can be improved.

The default method of boosting the result away from singularity is completely unstudied. For optimization it is wise to move away from singularity, just how to do that best seems like a research question.

Revision

This help was last revised 2014 March 09.

Author(s)

Burns Statistics

See Also

`fitted.statfacmodBurSt`, `var.shrink.eqcor`, `cov.wt`, `slideWeight`.

Examples

```
## Not run:
varian1 <- factor.model.stat(retmat)

varfac <- factor.model.stat(retmat, nfac=0, zero=TRUE, output="fact")

varian2 <- fitted(varfac) # get matrix from factor model

varian3 <- factor.model.stat(retmat, nfac=rep(c(5,3,1), c(20,40,1)))

## End(Not run)
```

`fitted.statfacmodBurSt` *Variance Matrix From Statistical Factor Model*

Description

Takes a statistical factor model object and produces the variance matrix that it represents.

Usage

```
## S3 method for class 'statfacmodBurSt'
fitted(object, output = "full", ...)
```

Arguments

`object`	an object of class `statfacmodBurSt`.
`output`	a character string that must partially match one of: `"full"`, `"systematic"`, `"specific"`.
`...`	not used, but allows methods for inheriting objects to have additional arguments.

Value

a numeric matrix of the variance represented by the factor model, or the systematic or specific variance portion of the model.

This has two additional attributes:

`number.of.factors`	the number of factors used in the model.
`timestamp`	the date and time at which the model was originally created.

Revision

This help was last revised 2012 February 12.

Author(s)

Burns Statistics

See Also

`factor.model.stat`.

Examples

```
## Not run:
varfac <- factor.model.stat(retmat, zero=TRUE, output="factor")
# perhaps modify loadings and uniquenesses for missing values
varian2 <- fitted(varfac) # get variance matrix from factor model

specif <- fitted(varfac, output="specific") # diagonal matrix

## End(Not run)
```

---

partial.rainbow *Create Palette Function for Part of Rainbow*

Description

Returns a function suitable as the `color.palette` argument to `filled.contour` that contains a specified portion of the rainbow.

Usage

```
partial.rainbow(start = 0, end = 0.35)
```

Arguments

`start`	a number giving where the colors should start. Valid numbers range from 0 (red) to 1 (also red).
`end`	a number giving where the colors should end. Valid numbers range from 0 (red) to 1 (also red).

Value

a function similar to `rainbow` but with the `start` and `end` arguments (possibly) changed.

Details

This function was made to facilitate the construction of contour plots in the `alpha.proxy` function, but is of general use.

Revision

This help was last revised 2010 January 05.

References

Burns, Patrick (2003). "Portfolio Sharpening". Working Paper, Burns Statistics http://www.burns-stat.com/.

See Also

rainbow, filled.contour, alpha.proxy.

Examples

```
ap1 <- alpha.proxy(cor.man=seq(0, .3, len=50),
vol.man=seq(.15, .25, leng=60), plot=FALSE)
filled.contour(ap1, color.palette=partial.rainbow(start=.05, end=.3))
```

---

slideWeight *Generate Time Weights Flexiibly*

Description

Returns a vector of weights between 1 and 0 (inclusive). The general form is: some recent weights are 1, some early weights are 0, and weights between these vary linearly. It is allowable for no points to be specified as 0 or 1.

Usage

```
slideWeight(n, fractions = c(0, 1), observations = NULL,
locations = NULL)
```

Arguments

n	the length of the result – the number of observations.
fractions	length 2 numeric vector giving the fraction of the way between 0 and n where the ends of the linear slide should be. There is no requirement that the fractions need to be between 0 and 1. This is ignored if observations or locations is given.
observations	length 2 numeric vector giving the number of observations that should be 1 and the number of observations that should have any weight. The smaller is taken to be the former and the larger as the latter.
locations	length 2 numeric vector giving the locations (subscripts) for the ends of the linear slide. The smaller number is the index of the last 0, the larger is the index of the last value smaller than 1.

Value

a numeric vector of length n with values within the range of 0 and 1 (inclusive). The values are non-decreasing – meaning put more weight on more recent observations.

Details

The arguments fractions, observations and locations are all to control where the ends of the weights strictly between 0 and 1 are. Only one of them is used. locations takes precedence. fractions is used if neither of the other two are given.

This is suitable to give as the weights argument to var.shrink.eqcor and factor.model.stat.

Revision

This help was last revised 2014 March 09.

See Also

var.shrink.eqcor, factor.model.stat.

Examples

```
# examples assume number of observations is 200

# all weights either 0 or 1
rollwin50 <- slideWeight(200, observations=c(50,50))

# 50 with full weight, 100 more with partial weight
swght50.150 <- slideWeight(200, observations=c(50, 150))

# approximately default weights of var.shrink.eqcor and factor.model.stat
lindec3 <- slideWeight(200, fractions=c(-1/2, 3/2))
```

---

threeDarr *Combine matrices into 3D array*

Description

Returns a three-dimensional array given one or more matrices and instructions on how to combine them.

Usage

```
threeDarr(..., rep = 1, union = TRUE, slicenames = NULL)
```

Arguments

...	one or more matrices.
rep	an integer saying how many times to replicate the slices.
union	logical value: if TRUE, then the union of row and column names is created in the output. If FALSE, then the intersection is done.
slicenames	a character vector with length equal to the number of slices in the resulting array.

Value

an array with the first two dimensions being the union or intersection of the first two dimensions of the input matrices, and third dimension equal to the number of input matrices times the number of replications.

Details

The full name of arguments `rep`, `union` and `slicenames` must be given (no abbreviations) because they come after the three-dots construct.

Revision

This help was last revised 2012 January 22.

See Also

More general functionality of this sort can be found in the `abind` package.

Examples

```
## Not run:
multiple.var <- threeDarr(var1, var2, slicenames=c("standardVar", "crashVar"))

reparr <- threeDarr(matrix1, rep=3)

## End(Not run)
```

`var.add.benchmark` *Expand a Variance Matrix to Include a Benchmark*

Description

Takes a variance matrix (or 3-dimensional array) and a vector of weights for the benchmark. An object like the input is returned which includes a new asset representing the benchmark.

Usage

```
var.add.benchmark(variance, benchmark.weights, name = "benchmark",
sum.to.one = TRUE)
```

Arguments

`variance`	required. A variance matrix or a three-dimensional array where each slice of the third dimension is a variance matrix. This must have dimnames which specify the assets, and it must include all of the assets named in `benchmark.weights`.
`benchmark.weights`	required. A named vector giving the weights of the constituents of the benchmark.
`name`	character string to be used as the asset name for the benchmark.
`sum.to.one`	logical vector stating whether to check and enforce that `benchmark.weights` sums to one.

Value

a matrix or array similar to the input `variance`, but including an additional asset.

Details

The (absolute value of) the weights ideally sum to one. You can give it weights that do not sum to one (perhaps they sum to 100 or the market cap of the benchmark), but you will get a warning that the weights are being adjusted.

If your "benchmark" is something that doesn't sum to 1 (such as portfolio weights minus benchmark weights), then set the `sum.to.one` argument to `FALSE`.

Revision

This help was last revised 2012 January 20.

See Also

`var.relative.benchmark`, `var.shrink.eqcor`, `factor.model.stat`, `threeDarr`.

Examples

```
## Not run:
varian.ben <- var.add.benchmark(varian, ftse.constituents, "ftse100")

## End(Not run)

var.orig <- array(c(400, 32, 24, 32, 64, 9.6, 24, 9.6, 144), c(3,3),
list(c("equities", "bonds", "commodities"),
c("equities", "bonds", "commodities")))
var.aa <- var.add.benchmark(var.orig, c(equities=.6, bonds=.4), "e60b40")
```

`var.relative.benchmark` *Transform a Variance Matrix to be Relative to a Benchmark*

Description

Returns a matrix (or 3-dimensional array) with dimensions one smaller than the input. The returned variance is relative to the benchmark.

Usage

```
var.relative.benchmark(variance, benchmark)
```

Arguments

variance	required. A variance matrix or a three-dimensional array where each slice of the third dimension is a variance matrix.
benchmark	required. A character string naming which asset in variance is to be used as the benchmark.

Value

a matrix or array similar to the input variance, but with one less asset and containing variances that are relative to that asset.

There is a call attribute which gives the command that created the object (and hence the benchmark to which the object is relative).

Revision

This help was last revised 2012 January 20.

See Also

var.add.benchmark, var.shrink.eqcor, factor.model.stat, threeDarr.

Examples

```
var.orig <- array(c(400, 32, 24, 32, 64, 9.6, 24, 9.6, 144), c(3,3),
list(c("equities", "bonds", "commodities"),
c("equities", "bonds", "commodities")))
var.aa <- var.add.benchmark(var.orig, c(equities=.6, bonds=.4), "e60b40")

var.rel <- var.relative.benchmark(var.aa, "e60b40")
```

---

var.shrink.eqcor *Ledoit-Wolf Shrinkage Variance Estimate*

Description

Returns a variance matrix that shrinks towards the equal correlation matrix – a Ledoit-Wolf estimate.

Usage

```
var.shrink.eqcor(x, weights = seq(0.5, 1.5, length = nt), shrink = NULL,
center = TRUE, vol.shrink = 0, sd.min = 20, quan.sd = 0.9,
tol = 1e-4, compatible = FALSE, verbose=2)
```

Arguments

x	required. A numeric matrix. The rows are observations and the columns are the variables. In finance, this will be a matrix of returns where the rows are times and the columns are assets. For the default value of weights the most recent observation should be the last row. The number of columns may exceed the number of rows, and missing values are accepted. A column may even have all missing values.
weights	a numeric vector giving the observation weights, or NULL. Equal weights can be specified with NULL or with a single positive number. Otherwise, this must be a vector of non-negative numbers that is as long as the number of observations (rows of x).
shrink	either NULL meaning the shrinkage will be estimated, or a single number meant to be in the range of zero to one.
center	either a single logical value, or a numeric vector as long as the number of columns of x. This gives the expected value of each column. A value of TRUE means the (weighted) mean of the columns of x is used. A value of FALSE means zero is used.
vol.shrink	a number between zero and one (inclusive) that says how much to shrink the standard deviations towards the mean standard deviation.
sd.min	a single integer giving the minimum number of observations needed in a column of x for the estimated standard deviation to be used.
quan.sd	a single number in the range of zero to one (inclusive) that says what quantile of the (used) standard deviations to use as the estimate of standard deviation for columns without enough (or any) data.
tol	a single number indicating how large the smallest eigenvalues of the answer should be. All eigenvalues are made to be no smaller than tol times the maximum eigenvalue.
compatible	a logical value. If TRUE, then the variance is scaled by the number of observations rather than one less than the number of observations, and there will be a warning if weights are not all the same. This makes the results compatible with Ledoit-Wolf computations and with the cov.shrink function in package tawny. This also makes the variances smaller, which is not necessarily a good thing.
verbose	a number indicating the level of warning messages desired. This currently controls only one warning: If at least 1, then a warning will be issued if all the values in x are non-negative. In finance this is an indication that prices rather than returns are input (an easy mistake to make).

Value

an estimate of the variance matrix of x. The sample variance is shrunk towards equal correlation.

This has two additional attributes:

shrink	the estimated or input amount of shrinkage towards the equal correlation matrix from the sample variance.
timedate	the date and time at which the computation was done.

Details

Time weights are quite helpful for estimating variances from returns. The default weighting seems to perform reasonably well over a range of situations. However, time weighting was not studied for this estimator.

Warning

The default value for `weights` assumes that the last row is the most recent observation and the first observation is the most ancient observation.

Research Issues

The method of handling missing values used in the function has not been studied (at all).

The method of boosting the result away from singularity is completely unstudied. For optimization it is wise to move away from singularity, just how to do that best seems like a research question. The method used boosts the smallest eigenvalues, it might be better to increase the diagonal.

Revision

This help was last revised 2014 March 09.

Author(s)

Burns Statistics

References

Olivier Ledoit and Michael Wolf (2004) "Honey, I shrunk the sample covariance matrix". The Journal of Portfolio Management, volume 30, number 4.

See Also

`factor.model.stat`, `cov.wt`, `slideWeight`.

Examples

```
## Not run:
var1 <- var.shrink.eqcor(return.matrix)

var.unweighted <- var.shrink.eqcor(return.matrix, weights=1)

## End(Not run)
```

Index