Package 'lgpr'

September 24, 2023

Title Longitudinal Gaussian Process Regression

Version 1.2.4

Description Interpretable nonparametric modeling of longitudinal data using additive Gaussian process regression. Contains functionality for inferring covariate effects and assessing covariate relevances. Models are specified using a convenient formula syntax, and can include shared, group-specific, non-stationary, heterogeneous and temporally uncertain effects. Bayesian inference for model parameters is performed using 'Stan'. The modeling approach and methods are described in detail in Timonen et al. (2021) <doi:10.1093/bioinformatics/btab021>.

License GPL (>= 3)

Encoding UTF-8

LazyData true

Biarch true

Depends R (>= 3.4.0), methods

Imports Rcpp (>= 0.12.0), RcppParallel (>= 5.0.2), RCurl (>= 1.98), rstan (>= 2.26.0), rstantools (>= 2.3.1), bayesplot (>= 1.7.0), MASS (>= 7.3-50), stats (>= 3.4), ggplot2 (>= 3.1.0), gridExtra (>= 0.3.0)

LinkingTo BH (>= 1.75.0-0), Rcpp (>= 1.0.6), RcppEigen (>= 0.3.3.9.1), RcppParallel (>= 5.0.2), rstan (>= 2.26.0), StanHeaders (>= 2.26.0)

SystemRequirements GNU make

NeedsCompilation yes

RoxygenNote 7.2.3

Suggests knitr, rmarkdown, testthat, covr

URL `https://github.com/jtimonen/lgpr`

BugReports `https://github.com/jtimonen/lgpr/issues`

VignetteBuilder knitr

Author Juho Timonen [aut, cre] (<`https://orcid.org/0000-0003-2341-6765`>), Andrew Johnson [ctb]

Maintainer Juho Timonen <juho.timonen@iki.fi>

Repository CRAN

Date/Publication 2023-09-24 06:50:02 UTC

R topics documented:

`lgpr-package` *The 'lgpr' package.*

Description

Interpretable nonparametric modeling of longitudinal data using additive Gaussian process regression. Contains functionality for inferring covariate effects and assessing covariate relevances. Models are specified using a convenient formula syntax, and can include shared, group-specific, non-stationary, heterogeneous and temporally uncertain effects. Bayesian inference for model parameters is performed using 'Stan' (`rstan`). The modeling approach and methods are described in detail in Timonen et al. (2021).

Core functions

Main functionality of the package consists of creating and fitting an additive GP model:

- `lgp`: Specify and fit an additive GP model with one command.
- `create_model`: Define an lgpmodel object.

- `sample_model`: Fit a model by sampling the posterior distribution of its parameters and create an lgpfit object.
- `pred`: Computing model predictions and inferred covariate effects after fitting a model.
- `relevances`: Assessing covariate relevances after fitting a model.
- `prior_pred`: Prior predictive sampling to check if your prior makes sense.

Visualization

- `plot_pred`: Plot model predictions.
- `plot_components`: Visualize inferred model components.
- `plot_draws`: Visualize parameter draws.
- `plot_data`: Visualize longitudinal data.

Data

The data that you wish to analyze with 'lgpr' should be in an R `data.frame` where columns correspond to measured variables and rows correspond to observations. Some functions that can help working with such data frames are:

- `new_x`: Creating new test points where the posterior distribution of any function component or sum of all components, or the posterior predictive distribution can be computed after model fitting.
- Other functions: `add_factor`, `add_factor_crossing`, `add_dis_age`, `adjusted_c_hat`.

Vignettes and tutorials

See `https://jtimonen.github.io/lgpr-usage/index.html`. The tutorials focus on code and use cases, whereas the Mathematical description of lgpr models vignette describes the statistical models and how they can be customized in 'lgpr'.

Citation

Run `citation("lgpr")` to get citation information.

Feedback

Bug reports, PRs, enhancement ideas or user experiences in general are welcome and appreciated. Create an issue in Github or email the author.

Author(s)

Juho Timonen (first.last at iki.fi)

References

1. Timonen, J. et al. (2021). *lgpr: an interpretable non-parametric method for inferring covariate effects from longitudinal data*. Bioinformatics, url.
2. Carpenter, B. et al. (2017). *Stan: A probabilistic programming language*. Journal of Statistical Software 76(1).

`add_dis_age` *Easily add the disease-related age variable to a data frame*

Description

Creates the disease-related age covariate vector based on the disease initiation times and adds it to the data frame

Usage

```
add_dis_age(data, t_init, id_var = "id", time_var = "age")
```

Arguments

`data`	the original data frame
`t_init`	A named vector containing the observed initiation or onset time for each individual. The names, i.e. `names(t_init)`, should specify the individual id.
`id_var`	name of the id variable in `data`
`time_var`	name of the time variable in `data`

Value

A data frame with one column added. The new column will be called `dis_age`. For controls, its value will be `NaN`.

See Also

Other data frame handling functions: `add_factor_crossing()`, `add_factor()`, `adjusted_c_hat()`, `new_x()`, `split()`

`add_factor` *Easily add a categorical covariate to a data frame*

Description

Easily add a categorical covariate to a data frame

Usage

```
add_factor(data, x, id_var = "id")
```

Arguments

`data`	the original data frame
`x`	A named vector containing the category for each individual. The names should specify the individual id.
`id_var`	name of the id variable in `data`

Value

A data frame with one column added. The new column will have same name as the variable passed as input `x`.

See Also

Other data frame handling functions: `add_dis_age()`, `add_factor_crossing()`, `adjusted_c_hat()`, `new_x()`, `split()`

`add_factor_crossing` *Add a crossing of two factors to a data frame*

Description

Add a crossing of two factors to a data frame

Usage

```
add_factor_crossing(data, fac1, fac2, new_name)
```

Arguments

`data`	a data frame
`fac1`	name of first factor, must be found in `df`
`fac2`	name of second factor, must be found in `df`
`new_name`	name of the new factor

Value

a data frame

See Also

Other data frame handling functions: `add_dis_age()`, `add_factor()`, `adjusted_c_hat()`, `new_x()`, `split()`

`adjusted_c_hat` *Set the GP mean vector, taking TMM or other normalization into account*

Description

Creates the `c_hat` input for `lgp`, so that it accounts for normalization between data points in the `"poisson"` or `"nb"` observation model

Usage

```
adjusted_c_hat(y, norm_factors)
```

Arguments

`y`	response variable, vector of length `n`
`norm_factors`	normalization factors, vector of length `n`

Value

a vector of length `n`, which can be used as the `c_hat` input to the `lgp` function

See Also

Other data frame handling functions: `add_dis_age()`, `add_factor_crossing()`, `add_factor()`, `new_x()`, `split()`

`apply_scaling` *Apply variable scaling*

Description

Apply variable scaling

Usage

```
apply_scaling(scaling, x, inverse = FALSE)
```

Arguments

`scaling`	an object of class lgpscaling
`x`	object to which apply the scaling (numeric)
`inverse`	whether scaling should be done in inverse direction

Value

a similar object as `x`

See Also

Other variable scaling functions: `create_scaling()`

`as_character` *Character representations of different formula objects*

Description

Character representations of different formula objects

Usage

```
## S4 method for signature 'lgpexpr'
as.character(x)

## S4 method for signature 'lgpterm'
as.character(x)

## S4 method for signature 'lgpformula'
as.character(x)
```

Arguments

`x`	an object of some S4 class

Value

a character representation of the object

`create_model` *Create a model*

Description

See the Mathematical description of lgpr models vignette for more information about the connection between different options and the created statistical model.

Usage

```
create_model(
  formula,
  data,
  likelihood = "gaussian",
  prior = NULL,
  c_hat = NULL,
  num_trials = NULL,
  options = NULL,
  prior_only = FALSE,
  verbose = FALSE,
  sample_f = !(likelihood == "gaussian")
)
```

Arguments

`formula`	The model formula, where • it must contain exatly one tilde (~), with response variable on the left-hand side and model terms on the right-hand side • terms are be separated by a plus (+) sign • all variables appearing in `formula` must be found in `data` See the "Model formula syntax" section below (`lgp`) for instructions on how to specify the model terms.
`data`	A `data.frame` where each column corresponds to one variable, and each row is one observation. Continuous covariates and the response variable must have type `"numeric"` and categorical covariates must have type `"factor"`. Missing values should be indicated with `NaN` or `NA`. The response variable cannot contain missing values. Column names should not contain trailing or leading underscores.
`likelihood`	Determines the observation model. Must be either `"gaussian"` (default), `"poisson"`, `"nb"` (negative binomial), `"binomial"` or `"bb"` (beta binomial).
`prior`	A named list, defining the prior distribution of model (hyper)parameters. See the "Defining priors" section below (`lgp`).
`c_hat`	The GP mean. This should only be given if `sample_f` is `TRUE`, otherwise the GP will always have zero mean. If `sample_f` is `TRUE`, the given `c_hat` can be a vector of length `dim(data)[1]`, or a real number defining a constant GP mean. If not specified and `sample_f` is `TRUE`, `c_hat` is set to • `c_hat` = `mean(y)`, if `likelihood` is `"gaussian"`, • `c_hat = log(mean(y))` if `likelihood` is `"poisson"` or `"nb"`, • `c_hat = log(p/(1-p))`, where `p` = `mean(y/num_trials)` if `likelihood` is `"binomial"` or `"bb"`, where `y` denotes the response variable measurements.
`num_trials`	This argument (number of trials) is only needed when likelihood is `"binomial"` or `"bb"`. Must have length one or equal to the number of data points. Setting `num_trials=1` and `likelihood="binomial"` corresponds to Bernoulli observation model.

`options`	A named list with the following possible fields: • `delta` Amount of added jitter to ensure positive definite covariance matrices. • `vm_params` Variance mask function parameters (numeric vector of length 2). If `options` is `NULL`, default options are used. The defaults are equivalent to `options = list(delta = 1e-8, vm_params = c(0.025, 1))`.
`prior_only`	Should likelihood be ignored? See also `sample_param_prior` which can be used for any lgpmodel, and whose runtime is independent of the number of observations.
`verbose`	Should some informative messages be printed?
`sample_f`	Determines if the latent function values are sampled (must be `TRUE` if likelihood is not `"gaussian"`). If this is `TRUE`, the response variable will be normalized to have zero mean and unit variance.

Value

An object of class lgpmodel, containing the Stan input created based on parsing the specified `formula`, `prior`, and other options.

See Also

Other main functions: `draw_pred()`, `get_draws()`, `lgp()`, `pred()`, `prior_pred()`, `sample_model()`

---

create_model.covs_and_comps

Parse the covariates and model components from given data and formula

Description

Parse the covariates and model components from given data and formula

Usage

```
create_model.covs_and_comps(data, model_formula, x_cont_scl, verbose)
```

Arguments

`data`	A `data.frame` where each column corresponds to one variable, and each row is one observation. Continuous covariates and the response variable must have type `"numeric"` and categorical covariates must have type `"factor"`. Missing values should be indicated with `NaN` or `NA`. The response variable cannot contain missing values. Column names should not contain trailing or leading underscores.
`model_formula`	an object of class lgpformula

`x_cont_scl`	Information on how to scale the continuous covariates. This can either be • an existing list of objects with class lgpscaling, or • `NA`, in which case such list is created by computing mean and standard deviation from `data`
`verbose`	Should some informative messages be printed?

Value

parsed input to Stan and covariate scaling, and other info

See Also

Other internal model creation functions: `create_model.formula()`, `create_model.likelihood()`, `create_model.prior()`

`create_model.formula` *Create a model formula*

Description

Checks if formula is in advanced format and translates if not.

Usage

```
create_model.formula(formula, data, verbose = FALSE)
```

Arguments

`formula`	The model formula, where • it must contain exatly one tilde (`~`), with response variable on the left-hand side and model terms on the right-hand side • terms are be separated by a plus (`+`) sign • all variables appearing in `formula` must be found in `data` See the "Model formula syntax" section below (`lgp`) for instructions on how to specify the model terms.
`data`	A `data.frame` where each column corresponds to one variable, and each row is one observation. Continuous covariates and the response variable must have type `"numeric"` and categorical covariates must have type `"factor"`. Missing values should be indicated with `NaN` or `NA`. The response variable cannot contain missing values. Column names should not contain trailing or leading underscores.
`verbose`	Should some informative messages be printed?

Value

an object of class lgpformula

See Also

Other internal model creation functions: `create_model.covs_and_comps()`, `create_model.likelihood()`, `create_model.prior()`

`create_model.likelihood`

Parse the response variable and its likelihood model

Description

Parse the response variable and its likelihood model

Usage

```
create_model.likelihood(
  data,
  likelihood,
  c_hat,
  num_trials,
  y_name,
  sample_f,
  verbose
)
```

Arguments

`data`	A `data.frame` where each column corresponds to one variable, and each row is one observation. Continuous covariates and the response variable must have type `"numeric"` and categorical covariates must have type `"factor"`. Missing values should be indicated with `NaN` or `NA`. The response variable cannot contain missing values. Column names should not contain trailing or leading underscores.
`likelihood`	Determines the observation model. Must be either `"gaussian"` (default), `"poisson"`, `"nb"` (negative binomial), `"binomial"` or `"bb"` (beta binomial).
`c_hat`	The GP mean. This should only be given if `sample_f` is `TRUE`, otherwise the GP will always have zero mean. If `sample_f` is `TRUE`, the given `c_hat` can be a vector of length `dim(data)[1]`, or a real number defining a constant GP mean. If not specified and `sample_f` is `TRUE`, `c_hat` is set to • `c_hat = mean(y)`, if `likelihood` is `"gaussian"`, • `c_hat = log(mean(y))` if `likelihood` is `"poisson"` or `"nb"`, • `c_hat = log(p/(1-p))`, where `p = mean(y/num_trials)` if `likelihood` is `"binomial"` or `"bb"`, where `y` denotes the response variable measurements.

`num_trials`	This argument (number of trials) is only needed when likelihood is `"binomial"` or `"bb"`. Must have length one or equal to the number of data points. Setting `num_trials=1` and `likelihood="binomial"` corresponds to Bernoulli observation model.
`y_name`	Name of response variable
`sample_f`	Determines if the latent function values are sampled (must be `TRUE` if likelihood is not `"gaussian"`). If this is `TRUE`, the response variable will be normalized to have zero mean and unit variance.
`verbose`	Should some informative messages be printed?

Value

a list of parsed options

See Also

Other internal model creation functions: `create_model.covs_and_comps()`, `create_model.formula()`, `create_model.prior()`

`create_model.options` *Parse the given modeling options*

Description

Parse the given modeling options

Usage

```
create_model.options(options, verbose)
```

Arguments

`options`	A named list with the following possible fields: • `delta` Amount of added jitter to ensure positive definite covariance matrices. • `vm_params` Variance mask function parameters (numeric vector of length 2). If `options` is `NULL`, default options are used. The defaults are equivalent to `options = list(delta = 1e-8, vm_params = c(0.025, 1))`.
`verbose`	Should some informative messages be printed?

Value

a named list of parsed options

`create_model.prior` *Parse given prior*

Description

Parse given prior

Usage

```
create_model.prior(prior, stan_input, verbose)
```

Arguments

`prior`	A named list, defining the prior distribution of model (hyper)parameters. See the "Defining priors" section below (`lgp`).
`stan_input`	a list of stan input fields
`verbose`	Should some informative messages be printed?

Value

a named list of parsed options

See Also

Other internal model creation functions: `create_model.covs_and_comps()`, `create_model.formula()`, `create_model.likelihood()`

`create_plot_df` *Helper function for plots*

Description

Helper function for plots

Usage

```
create_plot_df(object, x = "age", group_by = "id")
```

Arguments

`object`	model or fit
`x`	x-axis variable name
`group_by`	grouping variable name (use `NULL` for no grouping)

Value

a data frame

`create_scaling` *Create a standardizing transform*

Description

Create a standardizing transform

Usage

```
create_scaling(x, name)
```

Arguments

`x`	variable measurements (might contain `NA` or `NaN`)
`name`	variable name

Value

an object of class lgpscaling

See Also

Other variable scaling functions: `apply_scaling()`

`dinvgamma_stanlike` *Density and quantile functions of the inverse gamma distribution*

Description

Using the same parametrization as Stan. More info here.

Usage

```
dinvgamma_stanlike(x, alpha, beta, log = FALSE)

qinvgamma_stanlike(p, alpha, beta)
```

Arguments

`x`	point where to compute the density
`alpha`	positive real number
`beta`	positive real number
`log`	is log-scale used?
`p`	quantile (must be between 0 and 1)

Value

density/quantile value

See Also

Other functions related to the inverse-gamma distribution: `plot_invgamma()`, `priors`

`draw_pred` *Draw pseudo-observations from posterior or prior predictive distribution*

Description

Draw pseudo-observations from predictive distribution. If `pred` contains draws from the component posterior (prior) distributions, then the output is draws from the posterior (prior) predictive distribution. If `pred` is not specified, then whether output draws are from prior or posterior predictive distribution depends on whether `fit` is created using the `lgp` option `prior_only=TRUE` or not.

Usage

```
draw_pred(fit, pred = NULL)
```

Arguments

`fit`	An object of class lgpfit that has been created using the `lgp` option `sample_f=TRUE`.
`pred`	An object of class Prediction, containing draws of each model component. If `NULL`, this is obtained using `get_pred(fit)`.

Value

An array with shape SxP, where S is the number of draws that `pred` contains and P is the length of each function draw. Each row $s = 1, \ldots, S$ of the output is one vector drawn from the predictive distribution, given parameter draw s.

See Also

Other main functions: `create_model()`, `get_draws()`, `lgp()`, `pred()`, `prior_pred()`, `sample_model()`

`example_fit` *Quick way to create an example lgpfit, useful for debugging*

Description

Quick way to create an example lgpfit, useful for debugging

Usage

```
example_fit(
  formula = y ~ id + age + age | SEX + age | LOC,
  likelihood = "gaussian",
  chains = 1,
  iter = 30,
  num_indiv = 6,
  num_timepoints = 5,
  ...
)
```

Arguments

`formula`	model formula
`likelihood`	observation model
`chains`	number of chains to run
`iter`	number of iterations to run
`num_indiv`	number of individuals (data simulation)
`num_timepoints`	number of time points (data simulation)
`...`	additional arguments to `lgp`

Value

An lgpfit object created by fitting the example model.

`fit_summary` *Print a fit summary.*

Description

Print a fit summary.

Usage

```
fit_summary(fit, ignore_pars = c("f_latent", "eta", "teff_raw", "lp__"))
```

Arguments

`fit`	an object of class lgpfit
`ignore_pars`	parameters and generated quantities to ignore from output

Value

`object` invisibly.

GaussianPrediction-class

An S4 class to represent analytically computed predictive distributions (conditional on hyperparameters) of an additive GP model

Description

An S4 class to represent analytically computed predictive distributions (conditional on hyperparameters) of an additive GP model

Usage

```
## S4 method for signature 'GaussianPrediction'
show(object)

## S4 method for signature 'GaussianPrediction'
component_names(object)

## S4 method for signature 'GaussianPrediction'
num_components(object)

## S4 method for signature 'GaussianPrediction'
num_paramsets(object)

## S4 method for signature 'GaussianPrediction'
num_evalpoints(object)
```

Arguments

`object`	GaussianPrediction object for which to apply a class method.

Methods (by generic)

- `show(GaussianPrediction)`: Print a summary about the object.
- `component_names(GaussianPrediction)`: Get names of components.
- `num_components(GaussianPrediction)`: Get number of components.
- `num_paramsets(GaussianPrediction)`: Get number of parameter combinations (different parameter vectors) using which predictions were computed.
- `num_evalpoints(GaussianPrediction)`: Get number of points where predictions were computed.

Slots

`f_comp_mean` component means

`f_comp_std` component standard deviations

`f_mean` signal mean (on normalized scale)

`f_std` signal standard deviation (on normalized scale)

`y_mean` predictive mean (on original data scale)

`y_std` predictive standard deviation (on original data scale)

`x` a data frame of points (covariate values) where the function posteriors or predictive distributions have been evaluated

See Also

Prediction

`get_draws` *Extract parameter draws from lgpfit or stanfit*

Description

Uses `extract` with `permuted = FALSE` and `inc_warmup = FALSE`.

Usage

```
get_draws(object, draws = NULL, reduce = NULL, ...)
```

Arguments

`object`	An object of class lgpfit or `stanfit`.
`draws`	Indices of the parameter draws. `NULL` corresponds to all post-warmup draws.
`reduce`	Function used to reduce all parameter draws into one set of parameters. Ignored if `NULL`, or if `draws` is not `NULL`.
`...`	Additional arguments to `rstan::extract()`.

Value

The return value is always a 2-dimensional array of shape `num_param_sets` x `num_params`.

See Also

Other main functions: `create_model()`, `draw_pred()`, `lgp()`, `pred()`, `prior_pred()`, `sample_model()`

`get_pred` *Extract model predictions and function posteriors*

Description

NOTE: It is not recommended for users to call this. Use `pred` instead.

Usage

```
get_pred(fit, draws = NULL, reduce = NULL, verbose = TRUE)
```

Arguments

`fit`	An object of class lgpfit.
`draws`	Indices of parameter draws to use, or `NULL` to use all draws.
`reduce`	Reduction for parameters draws. Can be a function that is applied to reduce all parameter draws into one parameter set, or `NULL` (no reduction). Has no effect if `draws` is specified.
`verbose`	Should more information and a possible progress bar be printed?

Value

an object of class GaussianPrediction or Prediction

`kernel` *Compute a kernel matrix (covariance matrix)*

Description

These have `STAN_kernel_*` counterparts. These R versions are provided for reference and are not optimized for speed. These are used when generating simulated data, and not during model inference.

Usage

```
kernel_eq(x1, x2, alpha = 1, ell)

kernel_ns(x1, x2, alpha = 1, ell, a)

kernel_zerosum(x1, x2, M)

kernel_bin(x1, x2, pos_class = 0)

kernel_cat(x1, x2)
```

```
kernel_varmask(x1, x2, a, vm_params)

kernel_beta(beta, idx1_expand, idx2_expand)
```

Arguments

`x1`	vector of length n
`x2`	vector of length m
`alpha`	marginal std (default = 1)
`ell`	lengthscale
`a`	steepness of the warping function rise
`M`	number of categories
`pos_class`	binary (mask) kernel function has value one if both inputs have this value, other wise it is zero
`vm_params`	vector of two mask function parameters.
`beta`	a parameter vector (row vector) of length `N_cases`
`idx1_expand`	integer vector of length n
`idx2_expand`	integer vector of length m

Value

A matrix of size n x m.

Functions

- `kernel_eq()`: Uses the exponentiated quadratic kernel.
- `kernel_ns()`: Uses the non-stationary kernel (input warping + squared exponential).
- `kernel_zerosum()`: Uses the zero-sum kernel. Here, `x1` and `x2` must be integer vectors (integers denoting different categories). Returns a binary matrix.
- `kernel_bin()`: Uses the binary (mask) kernel. Here, `x1` and `x2` must be integer vectors (integers denoting different categories). Returns a binary matrix.
- `kernel_cat()`: Uses the categorical kernel. Here, `x1` and `x2` must be integer vectors (integers denoting different categories). Returns a binary matrix.
- `kernel_varmask()`: Computes variance mask multiplier matrix. `NaN`'s in `x1` and `x2` will be replaced by 0.
- `kernel_beta()`: Computes the heterogeneity multiplier matrix. *NOTE:* `idx_expand` needs to be given so that `idx_expand[j]-1` tells the index of the beta parameter that should be used for the jth observation. If observation j doesn't correspond to any beta parameter, then `idx_expand[j]` should be 1.

`KernelComputer-class` *An S4 class to represent input for kernel matrix computations*

Description

An S4 class to represent input for kernel matrix computations

Usage

```
## S4 method for signature 'KernelComputer'
show(object)

## S4 method for signature 'KernelComputer'
num_components(object)

## S4 method for signature 'KernelComputer'
num_evalpoints(object)

## S4 method for signature 'KernelComputer'
num_paramsets(object)

## S4 method for signature 'KernelComputer'
component_names(object)
```

Arguments

`object`	The object for which to call a class method.

Methods (by generic)

- `show(KernelComputer)`: Print a summary about the object.
- `num_components(KernelComputer)`: Get number of components.
- `num_evalpoints(KernelComputer)`: Get number of evaluation points.
- `num_paramsets(KernelComputer)`: Get number of parameter sets.
- `component_names(KernelComputer)`: Get component names.

Slots

`input` Common input (for example parameter values).

`K_input` Input for computing kernel matrices between data points (`N` x `N`). A list.

`Ks_input` Input for computing kernel matrices between data and output points (`P` x `N`). A list.

`Kss_input` Input for computing kernel matrices between output points (`P` x `P`). A list, empty if `full_covariance=FALSE`.

`comp_names` Component names (character vector).

`full_covariance` Boolean value determining if this can compute full predictive covariance matrices (or just marginal variance at each point).

`no_separate_output_points` Boolean value determining if `Ks_input` and `Kss_input` are the same thing. Using this knowledge can reduce unnecessary computations of kernel matrices.

`STREAM` external pointer (for calling 'Stan' functions)

`lgp` *Main function of the 'lgpr' package*

Description

Creates an additive Gaussian process model using `create_model` and fits it using `sample_model`. See the Mathematical description of lgpr models vignette for more information about the connection between different options and the created statistical model.

Usage

```
lgp(
  formula,
  data,
  likelihood = "gaussian",
  prior = NULL,
  c_hat = NULL,
  num_trials = NULL,
  options = NULL,
  prior_only = FALSE,
  verbose = FALSE,
  sample_f = !(likelihood == "gaussian"),
  quiet = FALSE,
  skip_postproc = sample_f,
  ...
)
```

Arguments

`formula`	The model formula, where • it must contain exatly one tilde (~), with response variable on the left-hand side and model terms on the right-hand side • terms are be separated by a plus (+) sign • all variables appearing in `formula` must be found in `data` See the "Model formula syntax" section below (`lgp`) for instructions on how to specify the model terms.
`data`	A `data.frame` where each column corresponds to one variable, and each row is one observation. Continuous covariates and the response variable must have type `"numeric"` and categorical covariates must have type `"factor"`. Missing

	values should be indicated with `NaN` or `NA`. The response variable cannot contain missing values. Column names should not contain trailing or leading underscores.
`likelihood`	Determines the observation model. Must be either `"gaussian"` (default), `"poisson"`, `"nb"` (negative binomial), `"binomial"` or `"bb"` (beta binomial).
`prior`	A named list, defining the prior distribution of model (hyper)parameters. See the "Defining priors" section below (`lgp`).
`c_hat`	The GP mean. This should only be given if `sample_f` is `TRUE`, otherwise the GP will always have zero mean. If `sample_f` is `TRUE`, the given `c_hat` can be a vector of length `dim(data)[1]`, or a real number defining a constant GP mean. If not specified and `sample_f` is `TRUE`, `c_hat` is set to • `c_hat = mean(y)`, if `likelihood` is `"gaussian"`, • `c_hat = log(mean(y))` if `likelihood` is `"poisson"` or `"nb"`, • `c_hat = log(p/(1-p))`, where `p = mean(y/num_trials)` if `likelihood` is `"binomial"` or `"bb"`, where `y` denotes the response variable measurements.
`num_trials`	This argument (number of trials) is only needed when likelihood is `"binomial"` or `"bb"`. Must have length one or equal to the number of data points. Setting `num_trials=1` and `likelihood="binomial"` corresponds to Bernoulli observation model.
`options`	A named list with the following possible fields: • `delta` Amount of added jitter to ensure positive definite covariance matrices. • `vm_params` Variance mask function parameters (numeric vector of length 2). If `options` is `NULL`, default options are used. The defaults are equivalent to `options = list(delta = 1e-8, vm_params = c(0.025, 1))`.
`prior_only`	Should likelihood be ignored? See also `sample_param_prior` which can be used for any lgpmodel, and whose runtime is independent of the number of observations.
`verbose`	Can messages be printed during model creation? Has no effect if `quiet=TRUE`.
`sample_f`	Determines if the latent function values are sampled (must be `TRUE` if likelihood is not `"gaussian"`). If this is `TRUE`, the response variable will be normalized to have zero mean and unit variance.
`quiet`	Should all output messages be suppressed? You need to set also `refresh=0` if you want to suppress also the progress update messages from `sampling`.
`skip_postproc`	Should all postprocessing be skipped? If this is `TRUE`, the returned lgpfit object will likely be much smaller (if `sample_f=FALSE`).
`...`	Optional arguments passed to `sampling` or `optimizing`.

Value

Returns an object of the S4 class lgpfit.

Model formula syntax

There are two ways to define the model formula:

1. Using a common `formula`-like syntax, like in `y ~ age + age|id + sex`. Terms can consist of a single variable, such as `age`, or an interaction of two variables, such as `age|id`. In single-variable terms, the variable can be either continuous (numeric) or categorical (factor), whereas in interaction terms the variable on the left-hand side of the vertical bar (`|`) has to be continuous and the one on the right-hand side has to be categorical. Formulae specified using this syntax are translated to the advanced format so that
    - single-variable terms become `gp(x)` if variable `x` is numeric and `zs(x)` if `x` is a factor
    - interaction terms `x|z` become `gp(x)*zs(z)`
2. Using the advanced syntax, like in `y ~ gp(age) + gp(age)*zs(id) + het(id)*gp_vm(disAge)`. This creates lgprhs objects, which consist of lgpterms, which consist of lgpexprs. This approach must be used if creating nonstationary, heterogeneous or temporally uncertain components.

Either one of the approaches should be used and they should not be mixed.

Defining priors

The `prior` argument must be a named list, like `list(alpha=student_t(4), wrp=igam(30,10))`. See examples in tutorials. Possible allowed names are

- `"alpha"` = component magnitude parameters
- `"ell"` = component lengthscale parameters
- `"wrp"` = input warping steepness parameters
- `"sigma"` = noise magnitude (Gaussian obs. model)
- `"phi"` = inv. overdispersion (negative binomial obs. model)
- `"gamma"` = overdispersion (beta-binomial obs. model)
- `"beta"` = heterogeneity parameters
- `"effect_time"` = uncertain effect time parameters
- `"effect_time_info"` = additional options for the above

See `priors` for functions that can be used to define the list elements. If a parameter of a model is not given in this list, a default prior will be used for it.

When to not use default priors

It is not recommended to use default priors blindly. Rather, priors should be specified according to the knowledge about the problem at hand, as in any Bayesian analysis. In `lgpr` this is especially important when

1. Using a non-Gaussian likelihood or otherwise setting `sample_f = TRUE`. In this case the response variable is not normalized, so the scale on which the data varies must be taken into account when defining priors of the signal magnitude parameters `alpha` and possible noise parameters (`sigma`, `phi`, `gamma`). Also it should be checked if `c_hat` is set in a sensible way.

2. Using a model that contains a `gp_ns(x)` or `gp_vm(x)` expression in its formula. In this case the corresponding covariate `x` is not normalized, and the prior for the input warping steepness parameter `wrp` must be set according to the expected width of the window in which the nonstationary effect of `x` occurs. By default, the width of this window is about 36, which has been set assuming that the unit of `x` is months.

See Also

Other main functions: `create_model()`, `draw_pred()`, `get_draws()`, `pred()`, `prior_pred()`, `sample_model()`

`lgpexpr-class` *An S4 class to represent an lgp expression*

Description

An S4 class to represent an lgp expression

Slots

`covariate`	name of a covariate
`fun`	function name

See Also

See `operations` for performing arithmetics on lgprhs, lgpterm and lgpexpr objects.

`lgpfit-class` *An S4 class to represent the output of the `lgp` function*

Description

An S4 class to represent the output of the `lgp` function

Usage

```
## S4 method for signature 'lgpfit'
show(object)

## S4 method for signature 'lgpfit'
component_names(object)

## S4 method for signature 'lgpfit'
num_components(object)

## S4 method for signature 'lgpfit'
```

```
postproc(object, verbose = TRUE)

## S4 method for signature 'lgpfit'
contains_postproc(object)

## S4 method for signature 'lgpfit'
clear_postproc(object)

## S4 method for signature 'lgpfit'
get_model(object)

## S4 method for signature 'lgpfit'
get_stanfit(object)

## S4 method for signature 'lgpfit'
is_f_sampled(object)

## S4 method for signature 'lgpfit,missing'
plot(x, y)
```

Arguments

`object`	The object for which to apply a class method.
`verbose`	Can the method print any messages?
`x`	an lgpfit object to visualize
`y`	unused argument

Methods (by generic)

- `show(lgpfit)`: Print information and summary about the fit object.
- `component_names(lgpfit)`: Get names of model components.
- `num_components(lgpfit)`: Get number of model components. Returns a positive integer.
- `postproc(lgpfit)`: Apply postprocessing. Returns an updated lgpfit object (copies data).
- `contains_postproc(lgpfit)`: Check if object contains postprocessing information.
- `clear_postproc(lgpfit)`: Returns an updated (copies data) lgpfit object without any postprocessing information.
- `get_model(lgpfit)`: Get the stored lgpmodel object. Various properties of the returned object can be accessed as explained in the documentation of lgpmodel.
- `get_stanfit(lgpfit)`: Get the stored `stanfit` object. Various properties of the returned object can be accessed or plotted as explained here or in the documentation of `stanfit`.
- `is_f_sampled(lgpfit)`: Determine if inference was done by sampling the latent signal `f` (and its components).
- `plot(x = lgpfit, y = missing)`: Visualize parameter draws using `plot_draws`.

Slots

`stan_fit` An object of class `stanfit`.

`model` An object of class `lgpmodel`.

`num_draws` Total number of parameter draws.

`postproc_results` A named list containing possible postprocessing results.

See Also

For extracting parameter draws, see `get_draws`, or the `rstan` methods for `stanfit` objects.

For more detailed plotting functions, see `plot_draws`, `plot_beta`, `plot_warp`, `plot_effect_times`

`lgpformula-class` *An S4 class to represent an lgp formula*

Description

An S4 class to represent an lgp formula

Slots

`terms` an object of class lgprhs

`y_name` name of the response variable

`call` original formula call

See Also

See `operations` for performing arithmetics on lgprhs, lgpterm and lgpexpr objects.

`lgpmodel-class` *An S4 class to represent an additive GP model*

Description

An S4 class to represent an additive GP model

Usage

```
## S4 method for signature 'lgpmodel'
show(object)

## S4 method for signature 'lgpmodel'
parameter_info(object, digits = 3)

## S4 method for signature 'lgpmodel'
component_info(object)

## S4 method for signature 'lgpmodel'
num_components(object)

## S4 method for signature 'lgpmodel'
covariate_info(object)

## S4 method for signature 'lgpmodel'
component_names(object)

## S4 method for signature 'lgpmodel'
is_f_sampled(object)
```

Arguments

`object`	The object for which to apply a class method.
`digits`	number of digits to show for floating point numbers

Methods (by generic)

- `show(lgpmodel)`: Print information and summary about the object. Returns `object` invisibly.
- `parameter_info(lgpmodel)`: Get a parameter summary (bounds and priors). Returns a `data.frame`.
- `component_info(lgpmodel)`: Get a data frame with information about each model component.
- `num_components(lgpmodel)`: Get number of model components. Returns a positive integer.
- `covariate_info(lgpmodel)`: Get covariate information.
- `component_names(lgpmodel)`: Get names of model components.
- `is_f_sampled(lgpmodel)`: Determine if inference of the model requires sampling the latent signal `f` (and its components).

Slots

`formula` An object of class lgpformula

`data` The original unmodified data.

`stan_input` The data to be given as input to `rstan::sampling`

`var_names` List of variable names grouped by type.

`var_scalings` A named list with fields

- `y` - Response variable normalization function and its inverse operation. Must be an lgpscaling object.
- `x_cont` - Continuous covariate normalization functions and their inverse operations. Must be a named list with each element is an lgpscaling object.

`var_info` A named list with fields

- `x_cat_levels` - Names of the levels of categorical covariates before converting from factor to numeric.

`info` Other info in text format.

`sample_f` Whether the signal `f` is sampled or marginalized.

`full_prior` Complete prior information.

`lgprhs-class` *An S4 class to represent the right-hand side of an lgp formula*

Description

An S4 class to represent the right-hand side of an lgp formula

Slots

`summands` a list of one or more lgpterms

See Also

See `operations` for performing arithmetics on lgprhs, lgpterm and lgpexpr objects.

`lgpscaling-class` *An S4 class to represent variable scaling*

Description

An S4 class to represent variable scaling

Slots

`loc` original location (mean)

`scale` original scale (standard deviation)

`var_name` variable name

`lgpsim-class` *An S4 class to represent a data set simulated using the additive GP formalism*

Description

An S4 class to represent a data set simulated using the additive GP formalism

Usage

```
## S4 method for signature 'lgpsim'
show(object)

## S4 method for signature 'lgpsim,missing'
plot(x, y, ...)
```

Arguments

`object`	an lgpsim object
`x`	an lgpsim object to plot
`y`	not used
`...`	optional arguments passed to `plot_sim`

Methods (by generic)

- `show(lgpsim)`: Show summary of object.
- `plot(x = lgpsim, y = missing)`: Plot the data and generating process. For more information see `plot_sim`.

Slots

`data` the actual data

`response` name of the response variable in the data

`components` the drawn function components

`kernel_matrices` the covariance matrices for each gp

`info` A list with fields

  - `par_ell` the used lengthscale parameters
  - `par_cont` the parameters used to generate the continuous covariates
  - `p_signal` signal proportion

`effect_times` A list with fields

  - `true` possible true effect times that generate the disease effect
  - `observed` possible observed effect times

`lgpterm-class` *An S4 class to represent one formula term*

Description

An S4 class to represent one formula term

Slots

`factors` a list of at most two lgpexprs

See Also

See `operations` for performing arithmetics on lgprhs, lgpterm and lgpexpr objects.

`model_summary` *Print a model summary.*

Description

Print a model summary.

Usage

```
model_summary(object, digits = 3)

param_summary(object, digits = 3)
```

Arguments

`object`	a model or fit
`digits`	number of digits to round floats to

Value

`object` invisibly.

`new_x` *Create test input points for prediction*

Description

Replaces a continuous variable `x` in the data frame, and possibly another continuous variable `x_ns` derived from it, with new values, for each level of a grouping factor (usually id)

Usage

```
new_x(data, x_values, group_by = "id", x = "age", x_ns = NULL)
```

Arguments

`data`	A data frame. Can also be an lgpfit or lgpmodel object, in which case data is extracted from it.
`x_values`	the values of `x` to set for each individual
`group_by`	name of the grouping variable, must be a factor in `data` (or use `group_by=NA` to create a dummy grouping factor which has only one value)
`x`	of the variable along which to extend, must be a numeric in `data`
`x_ns`	of a nonstationary variable derived from `x`, must be a numeric in `data`

Value

a data frame containing the following columns

- all factors in the original `data`
- `x`
- `x_ns` (unless it is NULL)

See Also

Other data frame handling functions: `add_dis_age()`, `add_factor_crossing()`, `add_factor()`, `adjusted_c_hat()`, `split()`

`operations` *Operations on formula terms and expressions*

Description

Operations on formula terms and expressions

Usage

```
## S4 method for signature 'lgprhs,lgprhs'
e1 + e2

## S4 method for signature 'lgpterm,lgpterm'
e1 + e2

## S4 method for signature 'lgprhs,lgpterm'
e1 + e2

## S4 method for signature 'lgpterm,lgpterm'
e1 * e2
```

Arguments

`e1`	The first sum, term or expression
`e2`	The second sum, term or expression

Value

The behaviour and return type depend on the types of `e1` and `e2`. You can

- sum (+) two lgprhs's to yield an lgprhs
- sum (+) two lgpterm's to yield an lgprhs
- sum (+) an lgprhs and an lgpterm to yield an lgprhs
- multiply (`*`) two lgpterm's to yield an lgpterm

`plot_api_c` *Plot a generated/fit model component*

Description

Data frames specified in arguments `df`, and `df_err` must have a format where

- The first column is the grouping factor (usually id).
- The second column is the x-axis variable (usually age).
- The third column is the coloring factor. If name of the third column is `NA`, coloring is not done.
- A column named `y` must contain the y-axis variable (not for `df_err`).
- A column named `lower` (`upper`) must contain the lower (upper) bound of error bar (only for `df_err`).
- The posterior draw using which the fit has been computed can be specified with a factor named `_draw_` (only for `df`).

Usage

```
plot_api_c(
  df,
  df_err = NULL,
  alpha = 1,
  alpha_err = 0.2,
  no_err = FALSE,
  no_line = FALSE
)
```

Arguments

`df`	a data frame
`df_err`	a data frame
`alpha`	line opacity
`alpha_err`	ribbon opacity
`no_err`	hide error bar even when it would normally be plotted?
`no_line`	hide line even when it would normally be plotted?

Value

A `ggplot` object.

See Also

Other internal plot API functions: `plot_api_g()`

`plot_api_g` *Plot longitudinal data and/or model fit so that each subject/group has their own panel*

Description

Data frames specified in arguments `df_data`, `df_signal`, `df_fit`, and `df_fit_err` must have a format where

- the first column is the grouping factor (usually id)
- the second column is the x-axis variable (usually age)
- a column named `y` must contain the y-axis variable (not for `df_fit_err`)
- a column named `lower` (`upper`) must contain the lower (upper) bound of error bar (only for `df_fit_err`)
- a column named `draw` must be a factor that specifies the posterior draw using which the fit has been computed (only for `df_fit`)

Usage

```
plot_api_g(
  df_data,
  df_signal = NULL,
  df = NULL,
  df_err = NULL,
  teff_signal = NULL,
  teff_obs = NULL,
  i_test = NULL,
  color_signal = color_palette(2)[1],
  color = color_palette(2)[2],
  color_err = colorset("red", "light_highlight"),
  color_vlines = colorset("gray", "mid_highlight"),
  alpha = 1,
  alpha_err = 0.5,
  nrow = NULL,
  ncol = NULL,
  y_transform = function(x) x
)
```

Arguments

`df_data`	A data frame containing the observations.
`df_signal`	A data frame containing the true signal. Omitted if `NULL`.
`df`	A data frame containing the model fit, or a list of data frames. The list version can be used for example so that each list element corresponds to the fit computed using one parameter draw. Omitted if `NULL`.

`df_err`	A data frame containing error bars. Omitted if `NULL`. Must be `NULL` if `df_fit` is a list.
`teff_signal`	A named vector containing true effect times used to generate the signal. Omitted if `NULL`.
`teff_obs`	A named vector containing observed effect times. Omitted if `NULL`.
`i_test`	Indices of test points.
`color_signal`	Line color for true signal.
`color`	Line color for model fit.
`color_err`	Color of the error ribbon.
`color_vlines`	Two line colors for vertical lines (true and obs. effect time).
`alpha`	Line opacity for model fit.
`alpha_err`	Opacity of the error ribbon.
`nrow`	number of rows, an argument for `facet_wrap`
`ncol`	number of columns, an argument for `facet_wrap`
`y_transform`	A function to be applied to the third column of `df_data`.

Value

A `ggplot` object.

See Also

Other internal plot API functions: `plot_api_c()`

`plot_components` *Visualize all model components*

Description

This calls `plot_f` for all model components.

Usage

```
plot_components(
  fit,
  pred = NULL,
  group_by = "id",
  t_name = "age",
  MULT_STD = 2,
  verbose = TRUE,
  draws = NULL,
  reduce = function(x) base::mean(x),
  color_by = NA,
  no_err = FALSE,
```

```
  ylim = NULL,
  draw = TRUE,
  nrow = NULL,
  ncol = NULL,
  gg_add = NULL,
  x = NULL,
  ...
)
```

Arguments

`fit`	An object of class lgpfit.
`pred`	An object of class GaussianPrediction or Prediction. If `pred=NULL`, the `pred` function is called with the given `reduce` and `draws` arguments.
`group_by`	name of the grouping variable (use `group_by=NA` to avoid grouping)
`t_name`	name of the x-axis variable
`MULT_STD`	a multiplier for standard deviation
`verbose`	Can this print any messages?
`draws`	Only has effect if `pred=NULL`.
`reduce`	Only has effect if `pred=NULL`.
`color_by`	Names of coloring factors. Can have length 1 or equal to the number of components. See the `color_by` argument of `plot_f`.
`no_err`	Should the error ribbons be skipped even though they otherwise would be shown? Can have length 1 or equal to number of components + 1. See the `no_err` argument of `plot_api_c`.
`ylim`	a vector of length 2 (upper and lower y-axis limits), or NULL
`draw`	if this is TRUE, the plot grid is drawn using `arrangeGrob`
`nrow`	number of grid rows
`ncol`	number of grid columns
`gg_add`	additional ggplot obejct to add to each plot
`x`	Deprecated argument. This is now taken from the `pred` object to ensure compatibility.
`...`	additional arguments to `plot_api_c`

Value

a list of ggplot objects invisibly

See Also

Other main plot functions: `plot_draws()`, `plot_pred()`

`plot_data` *Vizualizing longitudinal data*

Description

Vizualizing longitudinal data

Usage

```
plot_data(
  data,
  x_name = "age",
  y_name = "y",
  group_by = "id",
  facet_by = NULL,
  color_by = NULL,
  highlight = NULL,
  main = NULL,
  sub = NULL
)
```

Arguments

`data`	A data frame.
`x_name`	Name of x-axis variable.
`y_name`	Name of the y-axis variable.
`group_by`	Name of grouping variable (must be a factor).
`facet_by`	Name of the faceting variable (must be a factor).
`color_by`	Name of coloring variable (must be a factor).
`highlight`	Value of category of the `group_by` variable that is highlighted. Can only be used if `color_by` is `NULL`.
`main`	main plot title
`sub`	plot subtitle

Value

a `ggplot` object

`plot_draws` *Visualize the distribution of parameter draws*

Description

Visualize the distribution of parameter draws

Usage

```
plot_draws(
  fit,
  type = "intervals",
  regex_pars = c("alpha", "ell", "wrp", "sigma", "phi", "gamma"),
  ...
)

plot_beta(fit, type = "dens", verbose = TRUE, ...)

plot_warp(
  fit,
  num_points = 300,
  window_size = 48,
  color = colorset("red", "dark"),
  alpha = 0.5
)

plot_effect_times(fit, type = "areas", verbose = TRUE, ...)
```

Arguments

`fit`	an object of class lgpfit
`type`	plot type, allowed options are "intervals", "dens", "areas", and "trace"
`regex_pars`	regex for parameter names to plot
`...`	additional arguments for the `bayesplot` function `mcmc_intervals`, `mcmc_dens`, `mcmc_areas` or `mcmc_trace`
`verbose`	Can any output be printed?
`num_points`	number of plot points
`window_size`	width of time window
`color`	line color
`alpha`	line alpha

Value

a `ggplot` object or list of them

Functions

- `plot_draws()`: visualizes the distribution of any set of model parameters (defaults to kernel hyperparameters and possible observation model parameters)
- `plot_beta()`: visualizes the distribution of the individual-specific disease effect magnitude parameter draws
- `plot_warp()`: visualizes the input warping function for different draws of the warping steepness parameter
- `plot_effect_times()`: visualizes the input warping function for different parameter draws

See Also

Other main plot functions: `plot_components()`, `plot_pred()`

`plot_inputwarp` *Visualize input warping function with several steepness parameter values*

Description

Visualize input warping function with several steepness parameter values

Usage

```
plot_inputwarp(wrp, x, color = colorset("red", "dark"), alpha = 0.5)
```

Arguments

`wrp`	a vector of values of the warping steepness parameter
`x`	a vector of input values
`color`	line color
`alpha`	line alpha

Value

a ggplot object

`plot_invgamma` *Plot the inverse gamma-distribution pdf*

Description

Plot the inverse gamma-distribution pdf

Usage

```
plot_invgamma(
  alpha,
  beta,
  by = 0.01,
  log = FALSE,
  IQR = 0.95,
  return_quantiles = FALSE,
  linecolor = colorset("red", "dark"),
  fillcolor = colorset("red", "mid")
)
```

Arguments

`alpha`	positive real number
`beta`	positive real number
`by`	grid size
`log`	is log-scale used?
`IQR`	inter-quantile range width
`return_quantiles`	should this return a list
`linecolor`	line color
`fillcolor`	fill color

Value

a `ggplot` object

See Also

Other functions related to the inverse-gamma distribution: `dinvgamma_stanlike()`, `priors`

`plot_pred` *Visualizing model predictions or inferred covariate effects*

Description

- Function draws at data points can be visualized using `plot_pred`. If the `pred` argument is `NULL`, it is computed using the `pred` function with `x=NULL`.
- The total signal `f` or any of its additive components can be plotted using `plot_f`.

Usage

```
plot_pred(
  fit,
  pred = NULL,
  group_by = "id",
  t_name = "age",
  MULT_STD = 2,
  verbose = TRUE,
  draws = NULL,
  reduce = function(x) base::mean(x),
  x = NULL,
  ...
)

plot_f(
  fit,
  pred = NULL,
  group_by = "id",
  t_name = "age",
  MULT_STD = 2,
  verbose = TRUE,
  draws = NULL,
  reduce = function(x) base::mean(x),
  comp_idx = NULL,
  color_by = NA,
  x = NULL,
  ...
)
```

Arguments

`fit`	An object of class lgpfit.
`pred`	An object of class GaussianPrediction or Prediction. If `pred=NULL`, the `pred` function is called with the given `reduce` and `draws` arguments.
`group_by`	name of the grouping variable (use `group_by=NA` to avoid grouping)
`t_name`	name of the x-axis variable

`MULT_STD`	a multiplier for standard deviation
`verbose`	Can this print any messages?
`draws`	Only has effect if `pred=NULL`.
`reduce`	Only has effect if `pred=NULL`.
`x`	Deprecated argument. This is now taken from the `pred` object to ensure compatibility.
`...`	additional arguments to `plot_api_g` or `plot_api_c`
`comp_idx`	Index of component to plot. The total sum is plotted if this is `NULL`.
`color_by`	name of coloring factor

Value

a `ggplot` object

See Also

Other main plot functions: `plot_components()`, `plot_draws()`

`plot_sim` *Visualize an lgpsim object (simulated data)*

Description

Visualize an lgpsim object (simulated data)

Usage

```
plot_sim(
  simdata,
  group_by = "id",
  x_name = "age",
  h_name = "h",
  y_name = "y",
  comp_idx = NULL,
  color_by = NA,
  verbose = TRUE,
  ...
)
```

Arguments

`simdata`	an object of class lgpsim
`group_by`	grouping factor
`x_name`	name of x-axis variable
`h_name`	name of the signal in `simdata$components` ("h" or "f")
`y_name`	name of response variable
`comp_idx`	Possible index of a component to be shown. If this is NULL, the data and total signal are shown.
`color_by`	coloring factor
`verbose`	should some information be printed?
`...`	additional arguments to `plot_api_g` or `plot_api_c`

Value

a `ggplot` object

`ppc` *Graphical posterior predictive checks*

Description

Graphical posterior predictive checks

Usage

```
ppc(fit, data = NULL, fun = default_ppc_fun(fit), verbose = TRUE, ...)
```

Arguments

`fit`	An object of class lgpfit that can been created with `sample_f=TRUE`.
`data`	the original data frame (deprecated argument with no effect, now obtained from fit object)
`fun`	`bayesplot` function name
`verbose`	Can this print any messages?
`...`	additional arguments passed to the default `pp_check` method in `bayesplot`

Value

a `ggplot` object

See Also

Introduction to graphical posterior predictive checks: here. Prior predictive check can be done by calling `prior_pred` and then `bayesplot::pp_check()`.

`pred` *Posterior predictions and function posteriors*

Description

- If `fit` is for a model that marginalizes the latent signal `f` (i.e. `is_f_sampled(fit)` is `FALSE`), this computes the analytic conditional posterior distributions of each model component, their sum, and the conditional predictive distribution. All these are computed for each (hyper)parameter draw (defined by `draws`), or other parameter set (obtained by a reduction defined by `reduce`). Results are stored in a GaussianPrediction object which is then returned.
- If `fit` is for a model that samples the latent signal `f` (i.e. `is_f_sampled(fit)` is `TRUE`), this will extract these function samples, compute their sum, and a version of the sum `f` that is transformed through the inverse link function. If `x` is not `NULL`, the function draws are extrapolated to the points specified by `x` using kernel regression. Results are stored in a Prediction object which is then returned.

Usage

```
pred(
  fit,
  x = NULL,
  reduce = function(x) base::mean(x),
  draws = NULL,
  verbose = TRUE,
  STREAM = get_stream(),
  c_hat_pred = NULL,
  force = FALSE,
  debug_kc = FALSE
)
```

Arguments

`fit`	An object of class lgpfit.
`x`	A data frame of points where function posterior distributions and predictions should be computed or sampled. The function `new_x` provides an easy way to create it. If this is `NULL`, the data points are used.
`reduce`	Reduction for parameters draws. Can be a function that is applied to reduce all parameter draws into one parameter set, or `NULL` (no reduction). Has no effect if `draws` is specified.
`draws`	Indices of parameter draws to use, or `NULL` to use all draws.
`verbose`	Should more information and a possible progress bar be printed?
`STREAM`	External pointer. By default obtained with `rstan::get_stream()`.
`c_hat_pred`	This is only used if the latent signal `f` was sampled. This input contains the values added to the sum `f` before passing through inverse link function. Must be a vector with length equal to the number of prediction points. If original

	`c_hat` was constant, then `c_hat_pred` can be ignored, in which case this will by default use the same constant.
`force`	This is by default `FALSE` to prevent unintended large computations that might crash R or take forever. Set it to `TRUE` try computing no matter what.
`debug_kc`	If this is `TRUE`, this only returns a KernelComputer object that is created internally. Meant for debugging.

Value

An object of class GaussianPrediction or Prediction.

See Also

Other main functions: `create_model()`, `draw_pred()`, `get_draws()`, `lgp()`, `prior_pred()`, `sample_model()`

`Prediction-class` *An S4 class to represent prior or posterior draws from an additive function distribution.*

Description

An S4 class to represent prior or posterior draws from an additive function distribution.

Usage

```
## S4 method for signature 'Prediction'
show(object)

## S4 method for signature 'Prediction'
component_names(object)

## S4 method for signature 'Prediction'
num_components(object)

## S4 method for signature 'Prediction'
num_paramsets(object)

## S4 method for signature 'Prediction'
num_evalpoints(object)
```

Arguments

`object`	Prediction object for which to apply a class method.

Methods (by generic)

- `show(Prediction)`: Print a summary about the object.
- `component_names(Prediction)`: Get names of components.
- `num_components(Prediction)`: Get number of components.
- `num_paramsets(Prediction)`: Get number of parameter combinations (different parameter vectors) using which predictions were computed.
- `num_evalpoints(Prediction)`: Get number of points where predictions were computed.

Slots

`f_comp`	component draws
`f`	signal draws
`h`	predictions (signal draws + scaling factor `c_hat`, transformed through inverse link function)
`x`	a data frame of points (covariate values) where the functions/predictions have been evaluated/sampled
`extrapolated`	Boolean value telling if the function draws are original MCMC draws or if they have been created by extrapolating such draws.

See Also

GaussianPrediction

`priors` *Prior definitions*

Description

These use the same parametrizations as defined in the 'Stan' documentation. See the docs for gamma and inverse gamma distributions.

Usage

```
uniform(square = FALSE)

normal(mu, sigma, square = FALSE)

student_t(nu, square = FALSE)

gam(shape, inv_scale, square = FALSE)

igam(shape, scale, square = FALSE)

log_normal(mu, sigma, square = FALSE)

bet(a, b)
```

Arguments

`square`	is prior for a square-transformed parameter?
`mu`	mean
`sigma`	standard deviation
`nu`	degrees of freedom
`shape`	shape parameter (alpha)
`inv_scale`	inverse scale parameter (beta)
`scale`	scale parameter (beta)
`a`	shape parameter
`b`	shape parameter

Value

a named list

See Also

Other functions related to the inverse-gamma distribution: `dinvgamma_stanlike()`, `plot_invgamma()`

Examples

```
# Log-normal prior
log_normal(mu = 1, sigma = 1)

# Cauchy prior
student_t(nu = 1)

# Exponential prior with rate = 0.1
gam(shape = 1, inv_scale = 0.1)

# Create a similar priors as in LonGP (Cheng et al., 2019)
# Not recommended, because a lengthscale close to 0 is possible.
a <- log(1) - log(0.1)
log_normal(mu = 0, sigma = a / 2) # for continuous lengthscale
student_t(nu = 4) # for interaction lengthscale
igam(shape = 0.5, scale = 0.005, square = TRUE) # for sigma
```

`prior_pred` *Prior (predictive) sampling*

Description

These functions take an lgpmodel object, and

- `prior_pred` samples from the prior predictive distribution of the model
- `sample_param_prior` samples only its parameter prior using `sampling`

Usage

```
prior_pred(
  model,
  verbose = TRUE,
  quiet = FALSE,
  refresh = 0,
  STREAM = get_stream(),
  ...
)

sample_param_prior(model, verbose = TRUE, quiet = FALSE, ...)
```

Arguments

`model`	An object of class lgpmodel.
`verbose`	Should more information and a possible progress bar be printed?
`quiet`	This forces `verbose` to be `FALSE`. If you want to suppress also the output from Stan, give the additional argument `refresh=0`.
`refresh`	Argument for `sampling`.
`STREAM`	External pointer. By default obtained with `rstan::get_stream()`.
`...`	Additional arguments for `sampling`.

Value

- `prior_pred` returns a list with components
  - `y_draws`: A matrix containing the prior predictive draws as rows. Can be passed to `bayesplot::pp_check()` for graphical prior predictive checking.
  - `pred_draws`: an object of class Prediction, containing prior draws of each model component and their sum
  - `param_draws`: a `stanfit` object of prior parameter draws (obtained by calling `sample_param_prior` internally)
- `sample_param_prior` returns an object of class `stanfit`

See Also

Other main functions: `create_model()`, `draw_pred()`, `get_draws()`, `lgp()`, `pred()`, `sample_model()`

`prior_to_num` *Convert given prior to numeric format*

Description

Convert given prior to numeric format

Usage

```
prior_to_num(desc)
```

Arguments

`desc`	Prior description as a named list, containing fields • `dist` - Distribution name. Must be one of 'uniform', 'normal', 'student-t', 'gamma', 'inv-gamma', 'log-normal' (case-insensitive) • `square` - Is the prior for a square-transformed parameter. Other list fields are interpreted as hyperparameters.

Value

a named list of parsed options

`read_proteomics_data` *Function for reading the built-in proteomics data*

Description

Function for reading the built-in proteomics data

Usage

```
read_proteomics_data(parentDir = NULL, protein = NULL, verbose = TRUE)
```

Arguments

`parentDir`	Path to local parent directory for the data. If this is `NULL`, data is downloaded from `https://github.com/jtimonen/lgpr-usage/tree/master/data/proteomics`.
`protein`	Index or name of protein.
`verbose`	Can this print some output?

Value

a `data.frame`

`relevances` *Assess component relevances*

Description

Assess component relevances

Usage

```
relevances(fit, reduce = function(x) base::mean(x), verbose = TRUE, ...)
```

Arguments

`fit`	an object of class `lgpfit`
`reduce`	a function to apply to reduce the relevances given each parameter draw into one value
`verbose`	Can this print any messages?
`...`	currently has no effect

Value

a named vector with length equal to `num_comps + 1`

`s4_generics` *S4 generics for lgpfit, lgpmodel, and other objects*

Description

S4 generics for lgpfit, lgpmodel, and other objects

Usage

```
parameter_info(object, digits)

component_info(object)

covariate_info(object)

component_names(object)

get_model(object)

is_f_sampled(object)

get_stanfit(object)
```

```
postproc(object, ...)

contains_postproc(object)

clear_postproc(object)

num_paramsets(object)

num_evalpoints(object)

num_components(object)
```

Arguments

`object`	object for which to apply the generic
`digits`	number of digits to show
`...`	additional optional arguments to pass

Value

- `parameter_info` returns a data frame with one row for each parameter and columns for parameter name, parameter bounds, and the assigned prior
- `component_info` returns a data frame with one row for each model component, and columns encoding information about model components
- `covariate_info` returns a list with names `continuous` and `categorical`, with information about both continuous and categorical covariates
- `component_names` returns a character vector with component names
- `get_model` for lgpfit objects returns an lgpmodel
- `is_f_sampled` returns a logical value
- `get_stanfit` returns a `stanfit` (rstan)
- `postproc` applies postprocessing and returns an updated lgpfit
- `clear_postproc` removes postprocessing information and returns an updated lgpfit
- `num_paramsets`, `num_evalpoints` and `num_components` return an integer

Functions

- `parameter_info()`: Get parameter information (priors etc.).
- `component_info()`: Get component information.
- `covariate_info()`: Get covariate information.
- `component_names()`: Get component names.
- `get_model()`: Get lgpmodel object.
- `is_f_sampled()`: Determine if signal f is sampled or marginalized.
- `get_stanfit()`: Extract stanfit object.

- `postproc()`: Perform postprocessing.
- `contains_postproc()`: Determine if object contains postprocessing information.
- `clear_postproc()`: Clear postprocessing information (to reduce size of object).
- `num_paramsets()`: Get number of parameter sets.
- `num_evalpoints()`: Get number of points where posterior is evaluated.
- `num_components()`: Get number of model components.

See Also

To find out which methods have been implemented for which classes, see lgpfit, lgpmodel, Prediction and GaussianPrediction.

`sample_model` *Fitting a model*

Description

- `sample_model` takes an lgpmodel object and fits it using `sampling`.
- `optimize_model` takes an lgpmodel object and fits it using `optimizing`.

Usage

```
sample_model(
  model,
  verbose = TRUE,
  quiet = FALSE,
  skip_postproc = is_f_sampled(model),
  ...
)

optimize_model(model, ...)
```

Arguments

`model`	An object of class lgpmodel.
`verbose`	Can messages be printed?
`quiet`	Should all output messages be suppressed? You need to set also `refresh=0` if you want to suppress also the progress update messages from `sampling`.
`skip_postproc`	Should all postprocessing be skipped? If this is `TRUE`, the returned lgpfit object will likely be much smaller (if `sample_f=FALSE`).
`...`	Optional arguments passed to `sampling` or `optimizing`.

Value

- `sample_model` returns an object of class lgpfit containing the parameter draws, the original `model` object, and possible postprocessing results. See documentation of lgpfit for more information.
- `optimize_model` directly returns the list returned by `optimizing`. See its documentation for more information.

See Also

Other main functions: `create_model()`, `draw_pred()`, `get_draws()`, `lgp()`, `pred()`, `prior_pred()`

`select` *Select relevant components*

Description

- `select` performs strict selection, returning either `TRUE` or `FALSE` for each component.
- `select.integrate` is like `select`, but instead of a fixed threshold, computes probabilistic selection by integrating over a threshold density.
- `select_freq` performs the selection separately using each parameter draw and returns the frequency at which each component was selected.
- `select_freq.integrate` is like `select_freq`, but instead of a fixed threshold, computes probabilistic selection frequencies by integrating over a threshold density.

Usage

```
select(fit, reduce = function(x) base::mean(x), threshold = 0.95, ...)

select_freq(fit, threshold = 0.95, ...)

select.integrate(
  fit,
  reduce = function(x) base::mean(x),
  p = function(x) stats::dbeta(x, 100, 5),
  h = 0.01,
  verbose = TRUE,
  ...
)

select_freq.integrate(
  fit,
  p = function(x) stats::dbeta(x, 100, 5),
  h = 0.01,
  verbose = TRUE,
  ...
)
```

Arguments

`fit`	An object of class `lgpfit`.
`reduce`	The `reduce` argument for `relevances`.
`threshold`	Threshold for relevance sum. Must be a value between 0 and 1.
`...`	Additional arguments to `relevances`.
`p`	A threshold density over interval [0,1].
`h`	A discretization parameter for computing a quadrature.
`verbose`	Should this show a progress bar?

Value

See description.

`show` *Printing formula object info using the show generic*

Description

Printing formula object info using the show generic

Usage

```
## S4 method for signature 'lgpformula'
show(object)

## S4 method for signature 'lgprhs'
show(object)

## S4 method for signature 'lgpterm'
show(object)
```

Arguments

`object`	an object of some S4 class

Value

the object invisibly

`sim.create_f` *Simulate latent function components for longitudinal data analysis*

Description

Simulate latent function components for longitudinal data analysis

Usage

```
sim.create_f(
  X,
  covariates,
  relevances,
  lengthscales,
  X_affected,
  dis_fun,
  bin_kernel,
  steepness,
  vm_params,
  force_zeromean
)
```

Arguments

`X`	input data matrix (generated by `sim.create_x`)
`covariates`	Integer vector that defines the types of covariates (other than id and age). Different integers correspond to the following covariate types: • 0 = disease-related age • 1 = other continuous covariate • 2 = a categorical covariate that interacts with age • 3 = a categorical covariate that acts as a group offset • 4 = a categorical covariate that that acts as a group offset AND is restricted to have value 0 for controls and 1 for cases
`relevances`	Relative relevance of each component. Must have be a vector so that `length(relevances) = 2 + length(covariates)`. First two values define the relevance of the individual-specific age and shared age component, respectively.
`lengthscales`	A vector so that `length(lengthscales) = 2 + sum(covariates %in% c(0,1,2))`.
`X_affected`	which individuals are affected by the disease
`dis_fun`	A function or a string that defines the disease effect. If this is a function, that function is used to generate the effect. If `dis_fun` is "gp_vm" or "gp_ns", the disease component is drawn from a nonstationary GP prior ("vm" is the variance masked version of it).

`bin_kernel`	Should the binary kernel be used for categorical covariates? If this is `TRUE`, the effect will exist only for group 1.
`steepness`	Steepness of the input warping function. This is only used if the disease component is in the model.
`vm_params`	Parameters of the variance mask function. This is only needed if `useMaskedVarianceKernel = TRUE`.
`force_zeromean`	Should each component (excluding the disease age component) be forced to have a zero mean?

Value

a data frame FFF where one column corresponds to one additive component

---

`sim.create_x` *Create an input data frame X for simulated data*

Description

Create an input data frame X for simulated data

Usage

```
sim.create_x(
  N,
  covariates,
  names,
  n_categs,
  t_data,
  t_jitter,
  t_effect_range,
  continuous_info
)
```

Arguments

`N`	Number of individuals.
`covariates`	Integer vector that defines the types of covariates (other than id and age). If not given, only the id and age covariates are created. Different integers correspond to the following covariate types: • 0 = disease-related age • 1 = other continuous covariate • 2 = a categorical covariate that interacts with age • 3 = a categorical covariate that acts as a group offset • 4 = a categorical covariate that that acts as a group offset AND is restricted to have value 0 for controls and 1 for cases

`names`	Covariate names.
`n_categs`	An integer vector defining the number of categories for each categorical covariate, so that `length(n_categs)` equals to the number of 2's and 3's in the `covariates` vector.
`t_data`	Measurement times (same for each individual, unless `t_jitter > 0` in which case they are perturbed).
`t_jitter`	Standard deviation of the jitter added to the given measurement times.
`t_effect_range`	Time interval from which the disease effect times are sampled uniformly. Alternatively, This can any function that returns the (possibly randomly generated) real disease effect time for one individual.
`continuous_info`	Info for generating continuous covariates. Must be a list containing fields `lambda` and `mu`, which have length 3. The continuous covariates are generated so that `x <- sin(a*t + b) + c`, where • `t <- seq(0, 2*pi, length.out = k)` • `a <- mu[1] + lambda[1]*stats::runif(1)` • `b <- mu[2] + lambda[2]*stats::runif(1)` • `c <- mu[3] + lambda[3]*stats::runif(1)`

Value

a list

`sim.create_y` *Simulate noisy observations*

Description

Simulate noisy observations

Usage

```
sim.create_y(noise_type, f, snr, phi, gamma, N_trials)
```

Arguments

`noise_type`	Either "gaussian", "poisson", "nb" (negative binomial), "binomial", or "bb" (beta-binomial).
`f`	The underlying signal.
`snr`	The desired signal-to-noise ratio. This argument is valid only when `noise_type` is `"gaussian"`.
`phi`	The inverse overdispersion parameter for negative binomial data. The variance is `g + g^2/phi`.
`gamma`	The dispersion parameter for beta-binomial data.
`N_trials`	The number of trials parameter for binomial data.

Value

A list `out`, where

- `out$h` is `f` mapped through an inverse link function (times `N_trials` if `noise_type` is binomial or beta-binomial)
- `out$y` is the noisy response variable.

`sim.kernels` *Compute all kernel matrices when simulating data*

Description

Compute all kernel matrices when simulating data

Usage

```
sim.kernels(
  X,
  types,
  lengthscales,
  X_affected,
  bin_kernel,
  useMaskedVarianceKernel,
  steepness,
  vm_params
)
```

Arguments

`X`	covariates
`types`	vector of covariate types, so that • 1 = ID • 2 = age • 3 = diseaseAge • 4 = other continuous covariate • 5 = a categorical covariate that interacts with age • 6 = a categorical covariate that acts as an offset
`lengthscales`	vector of lengthscales
`X_affected`	which individuals are affected by the disease
`bin_kernel`	whether or not binary (mask) kernel should be used for categorical covariates (if not, the zerosum kernel is used)
`useMaskedVarianceKernel`	should the masked variance kernel be used for drawing the disease component
`steepness`	steepness of the input warping function
`vm_params`	parameters of the variance mask function

Value

a 3D array

`simulate_data` *Generate an artificial longitudinal data set*

Description

Generate an artificial longitudinal data set.

Usage

```
simulate_data(
  N,
  t_data,
  covariates = c(),
  names = NULL,
  relevances = c(1, 1, rep(1, length(covariates))),
  n_categs = rep(2, sum(covariates %in% c(2, 3))),
  t_jitter = 0,
  lengthscales = rep(12, 2 + sum(covariates %in% c(0, 1, 2))),
  f_var = 1,
  noise_type = "gaussian",
  snr = 3,
  phi = 1,
  gamma = 0.2,
  N_affected = round(N/2),
  t_effect_range = "auto",
  t_observed = "after_0",
  c_hat = 0,
  dis_fun = "gp_warp_vm",
  bin_kernel = FALSE,
  steepness = 0.5,
  vm_params = c(0.025, 1),
  continuous_info = list(mu = c(pi/8, pi, -0.5), lambda = c(pi/8, pi, 1)),
  N_trials = 1,
  force_zeromean = TRUE
)
```

Arguments

`N`	Number of individuals.
`t_data`	Measurement times (same for each individual, unless `t_jitter` > `0` in which case they are perturbed).
`covariates`	Integer vector that defines the types of covariates (other than id and age). If not given, only the id and age covariates are created. Different integers correspond to the following covariate types:

- 0 = disease-related age
- 1 = other continuous covariate
- 2 = a categorical covariate that interacts with age
- 3 = a categorical covariate that acts as a group offset
- 4 = a categorical covariate that that acts as a group offset AND is restricted to have value 0 for controls and 1 for cases

`names`	Covariate names.
`relevances`	Relative relevance of each component. Must have be a vector so that `length(relevances) = 2 + length(covariates)`. First two values define the relevance of the individual-specific age and shared age component, respectively.
`n_categs`	An integer vector defining the number of categories for each categorical covariate, so that `length(n_categs)` equals to the number of 2's and 3's in the `covariates` vector.
`t_jitter`	Standard deviation of the jitter added to the given measurement times.
`lengthscales`	A vector so that `length(lengthscales) = 2 + sum(covariates %in% c(0,1,2))`.
`f_var`	variance of f
`noise_type`	Either "gaussian", "poisson", "nb" (negative binomial), "binomial", or "bb" (beta-binomial).
`snr`	The desired signal-to-noise ratio. This argument is valid only when `noise_type` is `"gaussian"`.
`phi`	The inverse overdispersion parameter for negative binomial data. The variance is `g + g^2/phi`.
`gamma`	The dispersion parameter for beta-binomial data.
`N_affected`	Number of diseased individuals that are affected by the disease. This defaults to the number of diseased individuals. This argument can only be given if `covariates` contains a zero.
`t_effect_range`	Time interval from which the disease effect times are sampled uniformly. Alternatively, This can any function that returns the (possibly randomly generated) real disease effect time for one individual.
`t_observed`	Determines how the disease effect time is observed. This can be any function that takes the real disease effect time as an argument and returns the (possibly randomly generated) observed onset/initiation time. Alternatively, this can be a string of the form `"after_n"` or `"random_p"` or `"exact"`.
`c_hat`	a constant added to f
`dis_fun`	A function or a string that defines the disease effect. If this is a function, that function is used to generate the effect. If `dis_fun` is "gp_vm" or "gp_ns", the disease component is drawn from a nonstationary GP prior ("vm" is the variance masked version of it).
`bin_kernel`	Should the binary kernel be used for categorical covariates? If this is `TRUE`, the effect will exist only for group 1.

`steepness`	Steepness of the input warping function. This is only used if the disease component is in the model.
`vm_params`	Parameters of the variance mask function. This is only needed if `useMaskedVarianceKernel = TRUE`.
`continuous_info`	Info for generating continuous covariates. Must be a list containing fields `lambda` and `mu`, which have length 3. The continuous covariates are generated so that `x <- sin(a*t + b) + c`, where • `t <- seq(0, 2*pi, length.out = k)` • `a <- mu[1] + lambda[1]*stats::runif(1)` • `b <- mu[2] + lambda[2]*stats::runif(1)` • `c <- mu[3] + lambda[3]*stats::runif(1)`
`N_trials`	The number of trials parameter for binomial data.
`force_zeromean`	Should each component (excluding the disease age component) be forced to have a zero mean?

Value

An object of class lgpsim.

Examples

```
# Generate Gaussian data
dat <- simulate_data(N = 4, t_data = c(6, 12, 24, 36, 48), snr = 3)

# Generate negative binomially (NB) distributed count data
dat <- simulate_data(
  N = 6, t_data = seq(2, 10, by = 2), noise_type = "nb",
  phi = 2
)
```

`split` *Split data into training and test sets*

Description

- `split_by_factor` splits according to given factor
- `split_within_factor` splits according to given data point indices within the same level of a factor
- `split_within_factor_random` selects k points from each level of a factor uniformly at random as test data
- `split_random` splits uniformly at random
- `split_data` splits according to given data rows

Usage

```
split_by_factor(data, test, var_name = "id")

split_within_factor(data, idx_test, var_name = "id")

split_within_factor_random(data, k_test = 1, var_name = "id")

split_random(data, p_test = 0.2, n_test = NULL)

split_data(data, i_test, sort_ids = TRUE)
```

Arguments

`data`	a data frame
`test`	the levels of the factor that will be used as test data
`var_name`	name of a factor in the data
`idx_test`	indices point indices with the factor
`k_test`	desired number of test data points per each level of the factor
`p_test`	desired proportion of test data
`n_test`	desired number of test data points (if NULL, `p_test` is used to compute this)
`i_test`	test data row indices
`sort_ids`	should the test indices be sorted into increasing order

Value

a named list with names `train`, `test`, `i_train` and `i_test`

See Also

Other data frame handling functions: `add_dis_age()`, `add_factor_crossing()`, `add_factor()`, `adjusted_c_hat()`, `new_x()`

`testdata_001` *A very small artificial test data, used mostly for unit tests*

Description

A very small artificial test data, used mostly for unit tests

Usage

```
testdata_001
```

Format

A data frame with 24 rows and 6 variables:

id individual id, a factor with levels: 1, 2, 3, 4

age age

dis_age disease-related age

blood a continuous variable

sex a factor with 2 levels: Male, Female

y a continuous variable

See Also

Other built-in datasets: `testdata_002`

`testdata_002` *Medium-size artificial test data, used mostly for tutorials*

Description

Medium-size artificial test data, used mostly for tutorials

Usage

```
testdata_002
```

Format

A data frame with 96 rows and 6 variables:

id individual id, a factor with levels: 01-12

age age

diseaseAge disease-related age

sex a factor with 2 levels: Male, Female

group a factor with 2 levels: Case, Control

y a continuous variable

See Also

`read_proteomics_data`

Other built-in datasets: `testdata_001`

`validate` *Validate S4 class objects*

Description

Validate S4 class objects

Usage

```
validate_lgpexpr(object)

validate_lgpformula(object)

validate_lgpscaling(object)

validate_lgpfit(object)

validate_GaussianPrediction(object)

validate_Prediction(object)
```

Arguments

`object`	an object to validate

Value

`TRUE` if valid, otherwise reasons for invalidity

`var_mask` *Variance masking function*

Description

Variance masking function

Usage

```
var_mask(x, stp)
```

Arguments

`x`	a vector of length n
`stp`	a positive real number (steepness of mask function)

Value

a vector of length n

See Also

Other kernel utility functions: `warp_input()`

`warp_input` *Input warping function*

Description

Input warping function

Usage

```
warp_input(x, a)
```

Arguments

`x`	a vector of length n
`a`	steepness of the warping function rise

Value

a vector of warped inputs $w(x)$, length n

See Also

Other kernel utility functions: `var_mask()`

Index