Package 'hmer'

May 17, 2022

Type Package

Title History Matching and Emulation Package

Version 1.0.1

Maintainer Andrew Iskauskas <andrew.iskauskas@durham.ac.uk>

BugReports https://github.com/andy-iskauskas/hmer/issues

URL https://github.com/andy-iskauskas/hmer

Description A set of objects and functions for Bayes Linear emulation and history matching. Core functionality includes automated training of emulators to data, diagnostic functions to ensure suitability, and a variety of proposal methods for generating 'waves' of points. For details on the mathematical background, there are many papers available on the topic (see references attached to function help files); for details of the functions in this package, consult the manual or help files.

Suggests spelling, knitr, rmarkdown, deSolve, testthat (>= 3.0.0), covr

VignetteBuilder knitr

Depends R (>= 4.0.0)

Imports purrr, stringr, tidyr, ggplot2, lhs, MASS, R6, viridis, mvtnorm, GGally, rlang, isoband, mclust

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.2.0

Language en-GB

Config/testthat/edition 3

NeedsCompilation no

Author Andrew Iskauskas [aut, cre] (<https://orcid.org/0000-0003-2825-3651>)

Repository CRAN

Date/Publication 2022-05-17 14:50:02 UTC

R topics documented:

`analyze_diagnostic` *Diagnostic Analysis for Emulators*

Description

Produces summary and plots for diagnostics

Usage

```
analyze_diagnostic(
  in_data,
  output_name,
  targets = NULL,
  plt = interactive(),
  cutoff = 3,
  ...
)
```

Arguments

`in_data`	The data to perform the analysis on
`output_name`	The name of the output emulated
`targets`	If required or desired, the targets for the system outputs
`plt`	Whether or not to plot the analysis
`cutoff`	The implausibility cutoff for diagnostic 'ce'
`...`	Any other parameters to pass to subfunctions

Details

Given diagnostic information (almost certainly provided from `get_diagnostic`), we can plot the results and highlight the points that are worthy of concern or further consideration. Each diagnostic available has a plot associated with it which can be produced here:

Standardized Error: A histogram of standardized errors. Outliers should be considered, as well as whether very many points have either large or small errors.

Comparison Diagnostics: Error bars around points, corresponding to emulator prediction plus or minus emulator uncertainty. A green line indicates where the emulator and simulator prediction would be in complete agreement: error bars that do not overlap with this line (coloured red) are to be considered. Where targets are provided, the colouration is limited only to points where the simulator prediction would be close to the targets.

Classification Error: A point plot comparing emulator implausibility to simulator implausibility, sectioned into regions horizontally and vertically by `cutoff`. Points that lie in the lower right quadrant (i.e. emulator would reject; simulator would not) should be considered.

This function takes a data.frame that contains the input points, simulator values and, depending on the diagnostic, a set of summary measures. It returns a data.frame of any points that failed the diagnostic.

Value

A data.frame of failed points

References

Jackson (2018) <http://etheses.dur.ac.uk/12826>

See Also

`get_diagnostic`

Other diagnostic functions: `classification_diag()`, `comparison_diag()`, `get_diagnostic()`, `individual_errors()`, `residual_diag()`, `standard_errors()`, `summary_diag()`, `validation_diagnostics()`

`behaviour_plot` *Output Plotting*

Description

A simple diagnostic plot that compares the output values to input values, for each possible combination. If emulators are provided, the emulator predictions are plotted; otherwise the model outputs are plotted.

Usage

```
behaviour_plot(
  ems,
  points,
  model = missing(ems),
  out_names = unique(names(collect_emulators(ems))),
  targets = NULL
)
```

Arguments

`ems`	A set of `Emulator` objects.
`points`	A set of points at which to evaluate the emulator expectation
`model`	If TRUE, use the model outputs; else use emulator expectation
`out_names`	If no emulators are provided, use this argument to indicate outputs.
`targets`	If targets are provided, these are added into the plots.

Details

If emulators are provided, then the `points` argument is optional: if given then the emulator predictions will correspond to those at the points provided. If no points are provided, 100*d (where d is the number of input parameters) are sampled uniformly from the space and used to predict at.

If no emulators are provided, then points must be provided, along with the names of the outputs to plot; each named output must exist as a column in the points data.frame.

Value

The dependency plots.

See Also

Other visualisation tools: `diagnostic_wrap()`, `effect_strength()`, `emulator_plot()`, `output_plot()`, `plot_actives()`, `plot_lattice()`, `plot_wrap()`, `simulator_plot()`, `space_removed()`, `validation_pairs()`, `wave_dependencies()`, `wave_points()`, `wave_values()`

Examples

```
behaviour_plot(SIREmulators$ems, model = FALSE)
behaviour_plot(points = SIRSample$training, out_names = names(SIREmulators$ems))
#> Throws a warning
behaviour_plot(SIRMultiWaveEmulators, model = TRUE, targets = SIREmulators$targets)
```

`bimodal_emulator_from_data` *Bimodal Emulation*

Description

Performs emulation of bimodal outputs and/or systems.

Usage

```
bimodal_emulator_from_data(
  data,
  output_names,
  ranges,
  input_names = names(ranges),
  verbose = interactive(),
  ...
)
```

Arguments

`data`	The data to train emulators on (as in variance_emulator_from_data)
`output_names`	The names of the outputs to emulate
`ranges`	The parameter ranges
`input_names`	The names of the parameters (by default inferred from `ranges`)
`verbose`	Should status updates be provided?
`...`	Any other parameters to pass to emulator training

Details

In many stochastic systems, particularly disease models, the outputs exhibit bimodality - a familiar example is where a disease either takes off or dies out. In these cases, it is not sensible to emulate the outputs based on all realisations, and instead we should emulate each mode separately.

This function first tries to identify bimodality. If detected, it determines which of the outputs in the data exhibits the bimodality: to these two separate emulators are trained, one to each mode. The emulators are provided with any data that is relevant to their training; for example, bimodality can exist in some regions of parameter space but not others. Points where bimodality is present have their realisations allocated between the two modes while points where no bimodality exists have their realisations provided to both modes. Targets that do not exhibit bimodality are trained as a normal stochastic output: that is, using the default of `variance_emulator_from_data`.

The function also estimates the proportion of realisations in each mode for the set of outputs. This value is also emulated as a deterministic emulator and included in the output.

The output of the function is a list, containing three objects: `mode1`, `mode2`, and `prop`. The first two objects have the form produced by `variance_emulator_from_data` while `prop` has the form of an `emulator_from_data` output.

Value

A list `(mode1, mode2, prop)` of emulator lists and objects.

Examples

```
# Use the stochastic SIR dataset
SIR_ranges <- list(aSI = c(0.1, 0.8), aIR = c(0, 0.5), aSR = c(0, 0.05))
SIR_names <- c("I10", "I25", "I50", "R10", "R25", "R50")
b_ems <- bimodal_emulator_from_data(SIR_stochastic$training, SIR_names, SIR_ranges)
```

`BirthDeath` *Birth-Death Model Results*

Description

An RData object containing two data.frames. The first consists of ten parameter sets run through a simple, two-parameter, stochastic birth-death model; five of the points have 500 replicates and the other five have only 5 replicates. The second consists of ten further points, each with ten replicates. The objects are denoted `training` and `validation`, representing their expected usage.

Usage

```
BirthDeath
```

Format

A list of two data.frames `training` and `validation`: each data.frame has the following columns:

lambda Birth rate

mu Death rate

Y The number of people at time t = 15

Details

The initial population for the simulations is 100 people; the model is run until t = 15 to obtain the results to emulate.

`classification_diag` *Classification Diagnostics*

Description

Shorthand function for diagnostic test ‘ce’.

Usage

```
classification_diag(
  emulator,
  targets,
  validation,
  cutoff = 3,
  plt = interactive()
)
```

Arguments

`emulator`	The emulator in question
`targets`	The output targets
`validation`	The validation set
`cutoff`	The implausibility cutoff
`plt`	Whether to plot or not

Details

For details of the function, see `get_diagnostic` and for the plot see `analyze_diagnostic`.

Value

A data.frame of failed points

References

Jackson (2018) <http://etheses.dur.ac.uk/12826>

See Also

`get_diagnostic`, `analyze_diagnostic`, `validation_diagnostics`

Other diagnostic functions: `analyze_diagnostic()`, `comparison_diag()`, `get_diagnostic()`, `individual_errors()`, `residual_diag()`, `standard_errors()`, `summary_diag()`, `validation_diagnostics()`

`comparison_diag` *Comparison Diagnostics*

Description

Shorthand function for diagnostic test ‘cd’.

Usage

```
comparison_diag(emulator, targets, validation, sd = 3, plt = interactive())
```

Arguments

`emulator`	The emulator in question
`targets`	The output targets
`validation`	The validation set
`sd`	The range of uncertainty allowed
`plt`	Whether to plot or not

Details

For details of the function, see `get_diagnostic` and for the plot see `analyze_diagnostic`.

Value

A data.frame of failed points

References

Jackson (2018) <http://etheses.dur.ac.uk/12826>

See Also

`get_diagnostic`, `analyze_diagnostic`, `validation_diagnostics`

Other diagnostic functions: `analyze_diagnostic()`, `classification_diag()`, `get_diagnostic()`, `individual_errors()`, `residual_diag()`, `standard_errors()`, `summary_diag()`, `validation_diagnostics()`

`Correlator` *Correlation Structure*

Description

Creates a correlation structure, with the necessary specifications.

The correlator has three main elements: the type of correlator, the associated hyperparameters, and the nugget term. The nugget term is broadly separate from the other two parameters, being type-independent.

Constructor

`Correlator$new(corr, hp, nug)`

Arguments

`corr` The type of correlation function. This is provided as a string which corresponds exactly with a function - the function should take three arguments `x, xp, hp`. This gives a correlation function u(x, xp) defined by hyperparameters `hp`. For a simple example, see `exp_sq`.

`hp` The associated hyperparameters needed to define the correlation structure, as a named list. In the case of `exp_sq`, this is a list of one element, `list(theta)`.

`nug` The size of the nugget term. In situations where not all variables are active, the main part of u(x) operates only on the active parts, xA. The presence of the nugget term accounts for the fact that points at the same position in the active space need not be at the same position in the full space.

By default, `Correlator$new()` initialises with `corr = exp_sq`, `hp = list(theta = 0.1)`, and `nug = 0`.

Accessor Methods

`get_corr(x, xp = NULL, actives = TRUE)` Returns the correlation between two points. If `xp` is `NULL`, then this is correlation between a set of points and themselves (i.e. 1 on the diagonal). All variables are assumed to be active unless otherwise stated in `actives`.

`get_hyper_p()` Returns the list of hyperparameters.

`print()` Produces a summary of the correlation structure specification.

Object Methods

`set_hyper_p(hp, nugget)` Modifies the hyperparameter and/or nugget terms. Returns a new `Correlator` object.

Options for Correlations

The default choice (and that supported by other functions in this package, particularly emulator_from_data) for the correlation structure is exponential-squared, due to the useful properties it possesses. However, one can manually instantiate a Correlator with a different underlying structure. Built-in alternatives are as follows, as well as whether a form exists for its derivative:

`matern` the Matérn function (derivative exists)

`orn_uhl` the Ornstein-Uhlenbeck function (no derivative)

`rat_quad` the rational quadratic function (derivative exists)

One more function, `gamma_exp`, is available but not directly supported by `emulator_from_data`, for example, due to its very limited suitability to emulating model outputs. However, this can be used as a test case for writing one's own correlation functions and using them with `emulator_from_data`.

A user-defined correlation function can be provided to the Correlator: the requirements are that the function accept data.matrix objects as its first and second arguments, and accept a named list of hyperparameters as its third argument, and return a matrix of correlations between rows of the data.matrices. If a derivative also exists, it should take the same name as the correlation function with "_d" appended to it, and the directions to differentiate with respect to should come after the hyperparameter argument. For example, the rational quadratic functions have the form

`rat_quad(x1, x2, hp = list(alpha, theta))`

`rat_quad_d(x1, x2, hp = list(alpha, theta), dx1, dx2)`

If defining a custom correlation function, care should be taken with hyperparameter estimation - see `emulator_from_data` examples for details.

Examples

```
test_corr <- Correlator$new(nug = 0.1)
test_corr
point1 <- data.frame(a = 0.1, b = 0.2, c = 0.3)
point2 <- data.frame(a = 0.15, b = 0.18, c = 0.295)
test_corr$get_corr(point1) #> 1
test_corr$get_corr(point1, point2) #> 0.6717557
test_corr$get_corr(point1, point2, actives = c(TRUE, TRUE, FALSE)) #> 0.6734372
```

```
new_corr <- test_corr$set_hyper_p(list(theta = 0.5), nug = 0.01)
new_corr$get_corr(point1, point2) #> 0.9784845
new_corr$get_corr(point1, point2, actives = c(TRUE, TRUE, FALSE)) #> 0.9785824

mat_corr <- Correlator$new('matern', list(nu = 1.5, theta = 0.5))
mat_corr$get_corr(data.frame(a = c(1, 0.9), b = c(4, 4.2)))
```

`diagnostic_wrap` *Diagnostic plots for wave outputs*

Description

A wrapper function for the set of diagnostic plots for multiple waves.

Usage

```
diagnostic_wrap(
  waves,
  targets,
  output_names = names(targets),
  input_names = names(waves[[1]])[!names(waves[[1]]) %in% names(targets)],
  directory = NULL,
  s.heights = rep(1000, 4),
  s.widths = s.heights,
  include.norm = TRUE,
  include.log = TRUE,
  ...
)
```

Arguments

`waves`	The wave points, as a list of data.frames.
`targets`	The output targets.
`output_names`	The outputs to plot.
`input_names`	The inputs to plot.
`directory`	The location of files to be saved (if required).
`s.heights`	The heights of the saved pngs (if directory is not NULL).
`s.widths`	The widths of the saved pngs (if directory is not NULL).
`include.norm`	Should normalized versions of simulator_plot and wave_dependencies be made?
`include.log`	Should the log-scale version of simulator_plot be made?
`...`	Optional parameters (eg `p_size`, `l_wid`, ...)

Details

The functions `simulator_plot`, `wave_points`, `wave_points`, and `wave_dependencies` are called, one after the other, to allow diagnosis of waves of emulation.

The `directory` option should be used as follows. If the desired location is in fact a folder, it should end in "/"; if instead the structure requires each plot to be saved with a prefix, then it should be provided. For example, `directory = "Plots/"` in the first event or `directory = "Plots/unique-identifier"` in the second event.

Value

The set of plots (either into console or saved).

See Also

Other visualisation tools: `behaviour_plot()`, `effect_strength()`, `emulator_plot()`, `output_plot()`, `plot_actives()`, `plot_lattice()`, `plot_wrap()`, `simulator_plot()`, `space_removed()`, `validation_pairs()`, `wave_dependencies()`, `wave_points()`, `wave_values()`

Examples

```
diagnostic_wrap(SIRMultiWaveData, SIREmulators$targets)
diagnostic_wrap(SIRMultiWaveData, SIREmulators$targets,
 input_names = c('aSI', 'aIR'), output_names = c('nI', 'nR'),
 p_size = 0.8, l_wid = 0.8, wave_numbers = 1:3, zero_in = FALSE, surround = TRUE)
```

`directional_deriv` *Derivative inner product*

Description

Find the (uncertainty modified) inner product between the derivative at a point `x` and a proposed direction `v`.

Usage

```
directional_deriv(em, x, v, sd = NULL, ...)
```

Arguments

`em`	The emulator in question
`x`	The point in input space to evaluate at
`v`	The direction to assess
`sd`	How many standard deviations to consider.
`...`	Additional arguments to pass through (eg local.var to the emulator functions)

Details

Given a point `x` and a direction `v`, we find the overlap between E[f'(x)] and `v`. The emulated derivative has uncertainty associated with it: the variance is taken into account using $v^T Var[f'(x)]v$.

If `sd == NULL`, then only the (normed) overlap between the derivative and the direction vector is returned. Otherwise a pair of values are returned: these are the normed overlap plus or minus `sd` times the uncertainty.

This function is concerned with ascertaining whether a direction is oriented in the direction of the emulator gradient, subject to the uncertainty around the estimate of the derivative. It allows for a consideration of "emulated gradient descent".

Value

Either a single numeric or a pair of numerics (see description)

Examples

```
directional_deriv(SIREmulators$ems[[1]], SIRSample$validation[1,], c(1,1,1))
```

`directional_proposal` *Emulated Derivative Point Proposal*

Description

Proposes a new point by applying 'emulated gradient descent' on an existing point.

Usage

```
directional_proposal(
  ems,
  x,
  targets,
  accept = 2,
  hstart = 1e-04,
  hcutoff = 1e-09,
  iteration.measure = "exp",
  iteration.steps = 100,
  nv = 500
)
```

Arguments

`ems`	The emulators to evaluate with respect to.
`x`	The original point.
`targets`	The list of emulator targets.
`accept`	The implausibility below which we allow an output to worsen.

`hstart`	The initial step size.
`hcutoff`	The minimum allowed step size.
`iteration.measure`	Either ‘exp’ for expectation or ‘imp’ for implausibility.
`iteration.steps`	The number of allowed iterations.
`nv`	The number of directions on the n-sphere to try.

Details

Given a point (preferably close to the implausibility boundary) `x`, we can calculate the emulated gradient at this point for each emulator. If the estimate of the expectation at this point for a given emulator is larger than the target value, then we would like to move in the direction of greatest decrease for this emulator, and conversely for an estimate of the expectation that’s smaller than the target value. The combination of this information for every emulator under consideration defines a preferred set of directions of travel from this point.

We may try to find a shared direction which improves (or at least does not worsen) all emulator evaluations. If a point is already well inside the implausibility boundary for a given output (where ‘well inside’ is defined by the value of `accept`), we may allow this output to worsen in order to improve the others.

Provided a shared direction, v, can be identified, we iteratively move in this direction. Define the new proposed point x’ = x + h*v, where h is a step-size given by `hstart`. Compare the summary statistic (either expectational difference or implausibility) to that provided by the original point; if the new point gives improvement, then continue to move in this direction until no further improvement is possible for this step-size. The step-size is reduced (up to a minimum of `hcutoff`) and the process is repeated. Only finitely many iteration steps are permitted; this can be tuned by supplying a value of `iteration.steps`.

Value

Either a new proposal point, or the original point if an improvement could not be found.

Examples

```
# Take a point from the SIR system at later waves with low (but >3) implausibility
start_point <- SIRMultiWaveData[[2]][90,1:3]
ems <- SIRMultiWaveEmulators[[3]]
targs <- SIREmulators$targets
# Using expected error as measure
new_point1 <- directional_proposal(ems, start_point, targs)
# Using implausibility as measure
new_point2 <- directional_proposal(ems, start_point, targs, iteration.measure = 'imp')
all_points <- do.call('rbind.data.frame', list(start_point, new_point1, new_point2))
nth_implausible(ems, all_points, targs)
```

`effect_strength` *Find Effect Strength of Active Variables*

Description

Collates the linear and quadratic contributions of the active variables to the global emulators' behaviour

Usage

```
effect_strength(
  ems,
  plt = interactive(),
  line.plot = FALSE,
  grid.plot = FALSE,
  labels = TRUE,
  quadratic = TRUE,
  xvar = TRUE
)
```

Arguments

`ems`	The Emulator object(s) to be analysed.
`plt`	Should the results be plotted?
`line.plot`	Should a line plot be produced?
`grid.plot`	Should the effect strengths be plotted as a grid?
`labels`	Whether or not the legend should be included.
`quadratic`	Whether or not quadratic effect strength should be calculated.
`xvar`	Should the inputs be used on the x-axis?

Details

For a set of emulators, it can be useful to see the relative contributions of various parameters to the global part of the emulator (i.e. the regression surface). This function extracts the relevant information from a list of emulator objects.

The parameter `quadratic` controls whether quadratic effect strength is calculated and plotted (an unnecessary plot if, say, linear emulators have been trained). The remaining options control visual aspects of the plots: `line.plot` determines whether a line or bar (default) plot should be produced, `grid.plot` determines whether the results are plotted as a graph or a grid, and `labels` determines if a legend should be provided with the plot (for large numbers of emulators, it is advisable to set this to `FALSE`).

Value

A list of data.frames: the first is the linear strength, and the second quadratic.

See Also

Other visualisation tools: `behaviour_plot()`, `diagnostic_wrap()`, `emulator_plot()`, `output_plot()`, `plot_actives()`, `plot_lattice()`, `plot_wrap()`, `simulator_plot()`, `space_removed()`, `validation_pairs()`, `wave_dependencies()`, `wave_points()`, `wave_values()`

Examples

```
effect <- effect_strength(SIREmulators$ems)
effect_line <- effect_strength(SIREmulators$ems, line.plot = TRUE)
effect_grid <- effect_strength(SIREmulators$ems, grid.plot = TRUE)
```

`Emulator` *Bayes Linear Emulator*

Description

Creates a univariate emulator object.

The structure of the emulator is `f(x) = g(x) * beta + u(x)`, for regression functions `g(x)`, regression coefficients `beta`, and correlation structure `u(x)`. An emulator can be created with or without data; the preferred method is to create an emulator based on prior specifications in the absence of data, then use that emulator with data to generate a new one (see examples).

Constructor

`Emulator$new(basis_f, beta, u, ranges, ...)`

Arguments

Required:

`basis_f` A list of basis functions to be used. The constant function `function(x) 1` should be provided as the first element.

`beta` The specification for the regression parameters. This should be provided in the form `list(mu, sigma)`, where `mu` are the expectations of the coefficients (aligning with the ordering of `basis_f`) and `sigma` the corresponding covariance matrix.

`u` The specifications for the correlation structure. This should be specified in the form `list(sigma, corr)`, where `sigma` is a single-valued object, and `corr` is a Correlator object.

`ranges` A named list of ranges for the input parameters, provided as a named list of length-two numeric vectors.

Optional:

`data` A `data.frame` consisting of the data with which to adjust the emulator, consisting of input values for each parameter and the output.

`out_name` The name of the output variable.

`a_vars` A logical vector indicating which variables are active for this emulator.

`discs` Model discrepancies: does not include observational error. Ideally split into `list(internal = ..., external = ...)`.

Internal:

`model` If a linear model, or otherwise, has been fitted to the data, it lives here.

`original_em` If the emulator has been adjusted, the unadjusted `Emulator` object is stored, for use of `set_sigma` or similar.

`multiplier` A multiplicative factor to be applied to u_sigma. Typically equal to 1, unless changes have been made by, for example, `mult_sigma`.

Constructor Details

The constructor must take, as a minimum: a list of vectorised basis functions, whose length is equal to the number of regression coefficients; a correlation structure, which can be non-stationary; and the parameter ranges, used to scale all inputs to the range [-1,1].

The construction of a correlation structure is detailed in the documentation for Correlator.

Accessor Methods

`get_exp(x, include_c)` Returns the emulator expectation at a point, or at a collection of points. If `include_c = FALSE`, the contribution made by the correlation structure is not included.

`get_cov(x, xp = NULL, full = FALSE, include_c)` Returns the covariance between collections of points `x` and `xp`. If `xp` is not supplied, then this is equivalent to `get_cov(x, x, ...)`; if `full = TRUE`, then the full covariance matrix is calculated - this is FALSE by default due to most built-in uses requiring only the diagonal terms, and allows us to take advantage of computational tricks for efficiency.

`implausibility(x, z, cutoff = NULL)` Returns the implausibility for a collection of points `x`. The implausibility is the distance between the emulator expectation and a desired output value, weighted by the emulator variance and any external uncertainty. The target, z, should be specified as a named pair `list(val, sigma)`, or a single numeric value. If `cutoff = NULL`, the output is a numeric `I`; if `cutoff` is a numeric value, then the output is boolean corresponding to `I <= cutoff`.

`get_exp_d(x, p)` Returns the expectation of the derivative of the emulated function, E[f'(x)]. Similar in structure to `get_exp` but for the additional parameter `p`, which indicates which of the input dimensions the derivative is performed with respect to.

`get_cov_d(x, p1, xp = NULL, p2 = NULL, full = FALSE)` Returns the variance of the derivative of the emulated function, Var[f'(x)]. The arguments are similar to that of `get_cov`, but for the addition of parameters `p1` and `p2`, which indicate the derivative directions. Formally, the output of this function is equivalent to Cov[df/dp1, df/dp2].

`print(...)` Returns a summary of the emulator specifications.

`plot(...)` A wrapper for `emulator_plot` for a single Emulator object.

Object Methods

`adjust(data, out_name)` Performs Bayes Linear Adjustment, given `data`. The data should contain all input parameters, even inactive ones, and the single output that we wish to emulate. `adjust` creates a new `Emulator` object with the adjusted expectation and variance resulting from Bayes Linear adjustment, allowing for the requisite predictions to be made using `get_exp` and `get_cov`.

`set_sigma(sigma)` Modifies the (usually constant) global variance of the correlation structure, `Var[u(X)]`. If the emulator has been trained, the original emulator is modified and Bayes Linear adjustment is again performed.

`mult_sigma(m)` Modifies the global variance of the correlation structure via a multiplicative factor. As with `set_sigma`, this change will chain through any prior emulators if the emulator in question is Bayes Linear adjusted.

`set_hyperparams(hp, nugget)` Modifies the underlying correlator for `u(x)`. Behaves in a similar way to `set_sigma` as regards trained emulators. See the Correlator documentation for details of `hp` and `nugget`.

References

Goldstein & Wooff (2007) <ISBN: 9780470065662>

Craig, Goldstein, Seheult & Smith (1998) <doi:10.1111/1467-9884.00115>

Examples

```
basis_functions <- list(function(x) 1, function(x) x[[1]], function(x) x[[2]])
beta <- list(mu = c(1,2,3),
             sigma = matrix(c(0.5, -0.1, 0.2, -0.1, 1, 0, 0.2, 0, 1.5), nrow = 3))
u <- list(mu = function(x) 0, sigma = 3, corr = Correlator$new('exp_sq', list(theta = 0.1)))
ranges <- list(a = c(-0.5, 0.5), b = c(-1, 2))
em <- Emulator$new(basis_functions, beta, u, ranges)
em
# Individual evaluations of points
# Points should still be declared in a data.frame
em$get_exp(data.frame(a = 0.1, b = 0.1)) #> 0.6
em$get_cov(data.frame(a = 0.1, b = 0.1)) #> 9.5
# 4x4 grid of points
sample_points <- expand.grid(a = seq(-0.5, 0.5, length.out = 4), b = seq(-1, 2, length.out = 4))
em$get_exp(sample_points) # Returns 16 expectations
em$get_cov(sample_points) # Returns 16 variances
sample_points_2 <- expand.grid(a = seq(-0.5, 0.5, length.out = 3),
                               b = seq(-1, 2, length.out = 4))
em$get_cov(sample_points, xp = sample_points_2, full = TRUE) # Returns a 16x12 matrix of covariances


fake_data <- data.frame(a = runif(10, -0.5, 0.5), b = runif(10, -1, 2))
fake_data$c <- fake_data$a + 2*fake_data$b
newem <- em$adjust(fake_data, 'c')
all(round(newem$get_exp(fake_data[,names(ranges)]),5) == round(fake_data$c,5)) #>TRUE

matern_em <- Emulator$new(basis_f = c(function(x) 1, function(x) x[[1]], function(x) x[[2]]),
 beta = list(mu = c(1, 0.5, 2), sigma = diag(0, nrow = 3)),
 u = list(corr = Correlator$new('matern', list(nu = 1.5, theta = 0.4))),
 ranges = list(x = c(-1, 1), y = c(0, 3)))
matern_em$get_exp(data.frame(x = 0.4, y = 2.3))

newem_data <- Emulator$new(basis_functions, beta, u, ranges, data = fake_data)
all(round(newem$get_exp(fake_data[,names(ranges)]),5)
   == round(newem_data$get_exp(fake_data[,names(ranges)]), 5)) #>TRUE
```

```
newem$get_exp_d(sample_points, 'a')
newem$get_cov_d(sample_points, 'b', p2 = 'a')
```

`emulator_from_data` *Generate Emulators from Data*

Description

Given data from simulator runs, generates a set of univariate `Emulator` objects, one for each output.

Usage

```
emulator_from_data(
  input_data,
  output_names,
  ranges,
  input_names = names(ranges),
  beta,
  u,
  c_lengths,
  funcs,
  deltas,
  ev,
  quadratic = TRUE,
  beta.var = FALSE,
  adjusted = TRUE,
  discrepancies = NULL,
  has.hierarchy = FALSE,
  verbose = interactive(),
  na.rm = FALSE,
  check.ranges = FALSE,
  corr_name = "exp_sq",
  targets = NULL,
  ...
)
```

Arguments

`input_data`	Required. A data.frame containing parameter and output values.
`output_names`	Required. A character vector of output names.
`ranges`	A named list of input parameter ranges.
`input_names`	The names of the parameters (if `ranges` is not provided).
`beta`	A list of regression coefficients for each output.
`u`	A list of `Correlator` objects for each output.
`c_lengths`	A list of correlation lengths for each output.

`funcs`	A list of regression functions for each output.
`deltas`	Nugget terms for each correlation structure.
`ev`	Estimates of ensemble variability for each output.
`quadratic`	Should a quadratic or linear fit be found?
`beta.var`	Should regression coefficient uncertainty be included?
`adjusted`	Are the raw emulators wanted, or Bayes Linear updated ones?
`discrepancies`	Any internal or external discrepancies in the model.
`has.hierarchy`	For hierarchical emulators, this will be TRUE.
`verbose`	Should status updates be printed?
`na.rm`	If NAs exist in the dataset, should those rows be removed?
`check.ranges`	Should the ranges be modified in light of the data provided?
`corr_name`	What correlation function to use. Defaults to exp_sq
`targets`	If provided, outputs are checked for over/underestimation
`...`	Any additional parameters (eg for custom correlation functions)

Details

Many of the parameters that can be passed to this function are optional: the minimal operating example requires `input_data`, `output_names`, and one of `ranges` or `input_names`. If `ranges` is supplied, the input names are generated from that list, data.frame, or data.matrix; if only `input_names` is specified, then the ranges are assumed to be [-1, 1] for every input.

The ranges can be provided in alternative ways: either as a named list of length-2 numeric vectors (corresponding to the maximum and minimum for each parameter); as a data.frame with 2 columns where each row corresponds to a parameter; or as a data.matrix defined similarly as the data.frame. In the cases where the ranges are provided as a data.frame or a data.matrix, the `row.names` of the data object must be provided, corresponding to the names of the parameters.

If the minimum information is provided, then an emulator is fitted as follows. The basis functions and associated regression coefficients are generated using `step` and `lm` up to a desired order (default 2, determined by `quadratic`). These regression parameters are assumed to be ‘known’ unless `beta.var = TRUE`, in which case the derived parameter variance is taken from the model fit too (and the regression coefficients themselves can be modified by the maximum likelihood estimate performed below).

The correlation function c(x, x’) is assumed to be `exp_sq` and a corresponding `Correlator` object is created. The hyperparameters of the correlation structure are determined using a combination of maximum likelihood estimation and restriction to a ’sensible’ range of values, to avoid the correlation length tending to 0 or very large values. This determines the variance `sigma^2`, correlation length `theta`, any other hyperparameters (eg `nu` for the matern correlation function), and nugget term `delta`. The hyperparameter priors can be overridden either by directly specifying them using the `c_lengths` argument, or by supplying ranges to the `theta_ranges` argument. Examples of this customisation can be found in the examples to this function.

If `ev` is provided, then the ensemble variability is taken into account in the determination of the nugget term via a two-stage training process.

Some rudimentary data handling functionality is available but should be approached with caution. The `na.rm` option will strip out rows of the training data that have NA values in them; this of course may leave too few points to train to, and any consistent occurrence of NAs in model data should be investigated. The `check.ranges` option allows a redefinition of the ranges of the input parameters for emulator training; this is a common practice in later waves in order to maximise the predictive power of the emulators, but should only be used here if one is sure that the training set is representative of (and certainly spanning) the full minimum enclosing hyperrectangle.

Value

A list of `Emulator` objects.

Examples

```
# Use the SIRSample training dataset as an example.
ranges <- list(aSI = c(0.1, 0.8), aIR = c(0, 0.5), aSR = c(0, 0.05))
out_vars <- c('nS','nI','nR')
ems_linear <- emulator_from_data(SIRSample$training, out_vars, ranges, quadratic = FALSE)
ems_linear # Printout of the key information.



  ems_quad <- emulator_from_data(SIRSample$training, out_vars, ranges)
  ems_quad # Now includes quadratic terms (but only where needed)

 ems_unadjusted <- emulator_from_data(SIRSample$training, out_vars, ranges, adjusted = FALSE)
  ems_unadjusted # Looks the same as ems_quad, but the emulators are not BL adjusted

  # Reproduce the linear case, but with slightly changed beta values
  basis_f <- list(
   c(function(x) 1, function(x) x[[1]], function(x) x[[2]], function(x) x[[3]]),
   c(function(x) 1, function(x) x[[1]], function(x) x[[2]]),
   c(function(x) 1, function(x) x[[1]], function(x) x[[2]])
  )
  beta_vals <- list(
   list(mu = c(300, -260, 220, -120)),
   list(mu = c(120, 110, -260)),
   list(mu = c(580, 160, 130))
  )
  ems2 <- emulator_from_data(SIRSample$training, out_vars, ranges,
                             funcs = basis_f, beta = beta_vals)
  ems2
  # Custom correlation functions
  corr_structs <- list(
    list(sigma = 83, corr = Correlator$new('exp_sq', list(theta = 0.5), nug = 0.1)),
    list(sigma = 95, corr = Correlator$new('exp_sq', list(theta = 0.4), nug = 0.25)),
    list(sigma = 164, corr = Correlator$new('exp_sq', list(theta = 0.2), nug = 0.45))
  )
  ems3 <- emulator_from_data(SIRSample$training, out_vars, ranges,
                             u = corr_structs)
  # Using alternative correlation functions and c_lengths
  # Allow code to choose hyperparameters
  ems_matern <- emulator_from_data(SIRSample$training, out_vars, ranges,
                                   corr_name = 'matern')
```

```
# Providing hyperparameters to the function directly, via c_lengths
matern_hp <- list(list(theta = 0.8, nu = 1.5), list(theta = 0.6, nu = 2.5),
 list(theta = 1.2, nu = 0.5))
ems_matern2 <- emulator_from_data(SIRSample$training, out_vars, ranges,
                                  corr_name = 'matern', c_lengths = matern_hp)
# If only one set of hyperparameters are provided to c_lengths, they are used for all
ems_matern3 <- emulator_from_data(SIRSample$training, out_vars, ranges,
                                  corr_name = 'matern', c_lengths = matern_hp[[1]])
# "Custom" correlation function with user-specified ranges: gamma exponential
# 'gamma_exp' can be substituted for any correlation function - see Correlator documentation
ems_gamma <- emulator_from_data(SIRSample$training, out_vars, ranges,
                                  corr_name = 'gamma_exp',
                            theta_ranges = list(gamma = c(0.01, 2), theta = c(1/3, 2)))
```

`emulator_plot` *Plot Emulator Outputs*

Description

A function for plotting emulator expectations, variances, and implausibilities

Usage

```
emulator_plot(
  ems,
  plot_type = "exp",
  ppd = 30,
  targets = NULL,
  cb = FALSE,
  params = NULL,
  fixed_vals = NULL,
  nth = 1
)
```

Arguments

`ems`	An `Emulator` object, or a list thereof.
`plot_type`	The statistic to plot (see description or examples).
`ppd`	The number of points per plotting dimension
`targets`	If required, the targets from which to calculate implausibility
`cb`	A boolean representing whether a colourblind-friendly plot is produced.
`params`	Which two input parameters should be plotted?
`fixed_vals`	For fixed input parameters, the values they are held at.
`nth`	If plotting nth maximum implausibility, which level maximum to plot.

Details

Given a single emulator, or a set of emulators, the emulator statistics can be plotted across a two-dimensional slice of the parameter space. Which statistic is plotted is determined by `plot_type`: options are ‘exp’, ‘var’, ‘sd’, ‘imp’, and ‘nimp’, which correspond to expectation, variance, standard deviation, implausibility, and nth-max implausibility.

By default, the slice varies in the first two parameters of the emulators, and all other parameters are taken to be fixed at their mid-range values. This behaviour can be changed with the `params` and `fixed_vals` parameters (see examples).

If the statistic is ‘exp’, ‘var’ or ‘sd’, then the minimal set of parameters to pass to this function are `ems` (which can be a list of emulators or a single one) and `plot_type`. If the statistic is ‘imp’ or ‘nimp’, then the `targets` must be supplied - it is not necessary to specify the individual target for a single emulator plot. If the statistic is ‘nimp’, then the level of maximum implausibility can be chosen with the parameter `nth`.

Implausibility plots are typically coloured from green (low implausibility) to red (high implausibility): a colourblind-friendly option is available and can be turned on by setting `cb = TRUE`.

The granularity of the plot is controlled by the `ppd` parameter, determining the number of points per dimension in the grid. For higher detail, at the expense of longer computing time, increase this value. The default is 30.

Value

A ggplot object, or collection thereof.

See Also

Other visualisation tools: `behaviour_plot()`, `diagnostic_wrap()`, `effect_strength()`, `output_plot()`, `plot_actives()`, `plot_lattice()`, `plot_wrap()`, `simulator_plot()`, `space_removed()`, `validation_pairs()`, `wave_dependencies()`, `wave_points()`, `wave_values()`

Examples

```
# Reducing ppd to 10 for speed.
emulator_plot(SIREmulators$ems, ppd = 10)
emulator_plot(SIREmulators$ems$nS, ppd = 10)
emulator_plot(SIREmulators$ems, plot_type = 'var', ppd = 10, params = c('aIR', 'aSR'))

   emulator_plot(SIREmulators$ems, plot_type = 'imp', ppd = 10,
    targets = SIREmulators$targets,
    fixed_vals = list(aSR = 0.02))
   emulator_plot(SIREmulators$ems, plot_type = 'nimp', cb = TRUE,
    targets = SIREmulators$targets, nth = 2, ppd = 10)
```

`exp_sq` *Exponential squared correlation function*

Description

For points `x`, `xp` and a correlation length `theta`, gives the exponent of the squared distance between `x` and `xp`, weighted by `theta` squared.

Usage

```
exp_sq(x, xp, hp)
```

Arguments

`x`	A data.frame of rows corresponding to position vectors
`xp`	A data.frame of rows corresponding to position vectors
`hp`	The hyperparameter theta (correlation length)

Value

The exponential-squared correlation between x and xp.

References

Rasmussen & Williams (2005) <ISBN: 9780262182539>

Examples

```
exp_sq(data.frame(a=1), data.frame(a=2), list(theta = 0.1))
#> 3.720076e-44
exp_sq(data.frame(a=1,b=2,c=-1),data.frame(a=1.5,b=2.9,c=-0.7), list(theta = 0.2))
#> 3.266131e-13
```

`full_wave` *Automatic Wave Calculation*

Description

Performs a full wave of emulation and history matching, given data.

Usage

```
full_wave(
  data,
  ranges,
  targets,
  old_emulators = NULL,
  prop_train = 0.7,
  cutoff = 3,
  nth = 1,
  verbose = interactive(),
  ...
)
```

Arguments

`data`	The data to train with.
`ranges`	The ranges of the input parameters
`targets`	The output targets to match to.
`old_emulators`	Any emulators from previous waves.
`prop_train`	What proportion of the data is used for training.
`cutoff`	The implausibility cutoff for point generation and diagnostics.
`nth`	The level of maximum implausibility to consider.
`verbose`	Should progress be printed to console?
`...`	Any arguments to be passed to `emulator_from_data`.

Details

This function uses all of the functionality from the package in a relatively conservative form. The function performs the following steps:

1) Split the data into a training set and a validation set, where `prop_train` indicates what proportion of the data is used to train.

2) Perform emulator training using `emulator_from_data`. If a more involved specification is desired, optional arguments can be passed to `emulator_from_data` using the `...` argument.

3) Perform diagnostics on the trained emulators, removing emulators that do not display acceptable performance. Global emulator variance may also be modified to ensure that none of the emulators demonstrate misclassification errors (from `classification_diag`).

4) Ordering the remaining emulators from most restrictive to least restrictive on the dataset provided at this wave. Some point generation mechanisms terminate early if a point is ruled out by a single emulator, so the ordering ensures this happens earlier rather than later.

5) Generate the new points using the default method of `generate_new_runs`, using the normal procedure (for details, see the description for generate_new_runs).

If the parameter `old_emulators` is provided, this should be a list of emulators used at all previous waves - for example if `full_wave` is used to do a second wave of history matching, then `old_emulators` would contain the list of first-wave emulators.

The function returns a list of two objects: `emulators` corresponding to this wave's emulators, and `points` corresponding to the new proposed points. The points can then be put into the simulator to generate runs for a subsequent wave.

Value

A list of two objects: `points` and `emulators`

Examples

```
ranges <- list(aSI = c(0.1, 0.8), aIR = c(0, 0.5), aSR = c(0, 0.05))
default <- full_wave(do.call('rbind.data.frame', SIRSample), ranges,
 SIREmulators$targets)
non_quad <- full_wave(do.call('rbind.data.frame', SIRSample), ranges,
 SIREmulators$targets, quadratic = FALSE)
second <- full_wave(SIRMultiWaveData[[2]], ranges, SIREmulators$targets,
 old_emulators = SIRMultiWaveEmulators[[1]])
```

`gamma_exp` *Gamma-exponential correlation function*

Description

For points `x`, `xp`, and a pair of hyperparameters `gamma` and `theta`, gives the gamma-exponential correlation between the two points.

Usage

```
gamma_exp(x, xp, hp)
```

Arguments

`x`	A data.frame of rows corresponding to position vectors
`xp`	A data.frame of rows corresponding to position vectors
`hp`	The hyperparameters theta (correlation length) and gamma (exponent), as a named list

Details

The gamma-exponential correlation function, for d = |x-x'|, is given by $\exp(-(d/\theta)^\gamma)$. Gamma must be between 0 (exclusive) and 2 (inclusive).

Value

The gamma-exponential correlation between x and xp.

References

Rasmussen & Williams (2005) <ISBN: 9780262182539>

Examples

```
gamma_exp(data.frame(a=1), data.frame(a=2), list(gamma = 1.5, theta = 0.1))
#> 1.846727e-14
gamma_exp(data.frame(a=1,b=2,c=-1),data.frame(a=1.5,b=2.9,c=-0.7), list(gamma = 1.3, theta = 0.2))
#> 0.0001399953
```

`generate_new_runs` *Generate Proposal Points*

Description

Given a set of trained emulators, this finds the next set of points that will be informative for a subsequent wave of emulation.

Usage

```
generate_new_runs(
  ems,
  n_points,
  z,
  method = c("lhs", "line", "importance"),
  cutoff = 3,
  nth = 1,
  plausible_set,
  verbose = interactive(),
  cluster = FALSE,
  resample = 1,
  seek = 0,
  c_tol = 0.5,
  i_tol = 0.01,
  to_file = NULL,
  ...
)
```

Arguments

`ems`	A list of `Emulator` objects, trained on previous design points.
`n_points`	The desired number of points to propose for the next wave.
`z`	The targets to match to.
`method`	The method(s) to use.
`cutoff`	The implausibility cutoff(s) to compare outputs to.
`nth`	A parameter to be passed to the `n` argument of `nth_implausible`.

`plausible_set`	An optional set of known non-implausible points (for eg line sampling).
`verbose`	Should progress statements be printed to the console?
`cluster`	Should emulator clustering be considered in the LHS generation?
`resample`	Number of times to resample using line and/or importance sampling.
`seek`	How many ‘good’ points to search for
`c_tol`	The tolerance with which to determine that best implausibility has been reached.
`i_tol`	The tolerance on final desired implausibility
`to_file`	The filename to write to sequentially during proposal. Default is NULL (no writing)
`...`	Any parameters to pass to individual sampling functions, eg `distro` for importance sampling.

Details

If the method is `'lhs'`, a Latin hypercube is generated and non-implausible points from this design are retained. If enough points are accepted, the points outputted are chosen using either a maximin or V-optimality criteria (chosen by `measure.method`).

The methods `'line'` and `'importance'` both require a predetermined set of non-implausible points `s_points`; if they are not provided then lhs sampling is performed first.

The method `'line'` performs line sampling for boundary detection. Given a set of non-implausible points, rays are drawn between pairs of points (selected so as to maximise the distance between them), and more points are sampled along these lines. Points are kept if they lie near a boundary of the non-implausible space.

The method `'importance'` performs importance sampling, using a mixture distribution of multivariate normal or uniform spherical proposals around the current non-implausible points (determined by `distro`). The optimal standard deviation or radius is found using a burn-in phase, before the full set of points is generated.

The method `'slice'` performs slice sampling. Given one known non-implausible point, it attempts to find a minimum enclosing hyperrectangle for the non-implausible region around this point and samples uniformly from this, shrinking the hyperrectangle as appropriate. This method is also called if LH sampling has only generated one point (since any later methods require at least two points to be useful).

The method `'optical'` uses optical depth sampling: given a set of known non-implausible points, the an approximation of the one-dimensional marginal distributions in each parameter direction can be determined. From these derived marginals, points are sampled.

For any sampling strategy, the parameters `ems` and `z` must be provided.

The option `seek` determines how many points should be chosen that have a higher probability of matching targets, as opposed to not missing targets. Due to the danger of such an approach in terms of obtaining a representative space-filling design over the space, this value should not be too high: a rough guide is that it should be no larger than 10% of the desired number of points. The default is `seek = 0`.

The default behaviour is as follows. A set of initial points are generated from an LHD; line sampling is performed to find the boundaries; and finally this collection of points is augmented to the desired number of points by importance sampling using uniform spherical proposals.

In regions where the non-implausible space (at the given cutoff) is very hard to find, the function will start at a higher implausibility where it can find a space-filling design; using this as a starting point any other methods are performed. From this new proposal, a subset of lower-implausibility points are selected. This process iterates until either the desired implausibility has been reached or the process has reached a barrier to further reductions in implausibility. The argument `c_tol` is used to determine if the improvement in implausibility is small enough to justify stopping the process; by default this is 0.1. The process will also stop if it has produced points 'close to' the desired implausibility: the level of closeness is defined using the `i_tol` argument.

These methods may not work, or may work slowly, if the target space is very small compared to the current not-yet-ruled-out space, or it may miss small disconnected regions of parameter space.

Value

A data.frame containing the set of new points to run the model at.

Examples

```
# A simple example that uses a number of the native and ... parameter options
pts <- generate_new_runs(SIREmulators$ems, 100, SIREmulators$targets,
measure.method = 'maximin', distro = 'sphere', resample = 0)
pts_optical <- generate_new_runs(SIREmulators$ems, 100, SIREmulators$targets,
 method = c('optical'))
pts_slice <- generate_new_runs(SIREmulators$ems, 100, SIREmulators$targets,
 method = c('slice'))
pts_no_importance <- generate_new_runs(SIREmulators$ems, 100, SIREmulators$targets,
 method = c('line'))
```

`get_diagnostic` *Diagnostic Tests for Emulators*

Description

Given an emulator, return a diagnostic measure.

Usage

```
get_diagnostic(
  emulator,
  targets = NULL,
  validation = NULL,
  which_diag = "cd",
  stdev = 3,
  cleaned = NULL,
  warn = TRUE,
  kfold = NULL,
  ...
)
```

Arguments

`emulator`	An object of class Emulator
`targets`	If desired, the target values for the output(s) of the system
`validation`	If provided, the emulator is tested against the outputs of these points
`which_diag`	Which diagnostic measure to use (choosing from cd, ce, se above)
`stdev`	For ‘cd’, a measure of the allowed distance from prediction and reality
`cleaned`	Internal for stochastic emulators
`warn`	Should a warning be shown if ce is chosen and no targets provided?
`kfold`	Mainly internal: pre-computed k-fold diagnostic results for output
`...`	Any other parameters to be passed through to subfunctions.

Details

An emulator’s suitability can be checked in a number of ways. This function combines all current diagnostics available in the package, returning a context-dependent data.frame containing the results.

Comparison Diagnostics (cd): Given a set of points, the emulator expectation and variance are calculated. This gives a predictive range for the input point, according to the emulator. We compare this against the actual value given by the simulator: points whose emulator prediction is further away from the simulator prediction are to be investigated. This ’distance’ is given by `stdev`, and an emulator prediction correspondingly should not be further away from the simulator value than stdev*uncertainty.

Classification Error (ce): Given a set of targets, the emulator can determine implausibility of a point with respect to the relevant target, accepting or rejecting it as appropriate. We can define a similar ‘implausibility’ function for the simulator: the combination of the two rejection schemes gives four classifications of points. Any point where the emulator would reject the point but the simulator would not should be investigated.

Standardized Error (se): The known value at a point, combined with the emulator expectation and uncertainty, can be combined to provide a standardized error for a point. This error should not be too large, in general. but the diagnostic is more useful when looking at a collection of such measures, where systematic bias or over/underconfidence can be seen.

Which of the diagnostics is performed can be controlled by the `which_diag` argument. If performing classification error diagnostics, a set of targets must be provided; for all diagnostics, a validation (or holdout) set can be provided. If no such set is given, then the emulator diagnostics are performed with respect to its training points, using k-fold cross-validation.

Value

A data.frame consisting of the input points, output values, and diagnostic measures.

See Also

validation_diagnostics

Other diagnostic functions: `analyze_diagnostic()`, `classification_diag()`, `comparison_diag()`, `individual_errors()`, `residual_diag()`, `standard_errors()`, `summary_diag()`, `validation_diagnostics()`

Examples

```
# Use the simple SIR model via SIREmulators
get_diagnostic(SIREmulators$ems$nS, validation = SIRSample$validation)
# Classification error fails without the set of targets
get_diagnostic(SIREmulators$ems$nI, SIREmulators$targets, SIRSample$validation, 'ce')
# No validation set: k-fold cross-validation will be used.
get_diagnostic(SIREmulators$ems$nR, which_diag = 'se')
```

`HierarchicalEmulator` *Hierarchical Bayes Linear Emulator*

Description

Creates a univariate emulator with hierarchical structure.

This object does not differ extensively from the standard `Emulator` object, so most of the functionality will not be listed here: the main difference is that it allows for the variance structure of the emulator to be modified by a higher order object. The typical usage is to create a variance emulator, whose predictions inform the behaviour of a mean emulator with regard to a stochastic process.

Constructor

`HierarchicalEmulator$new(basis_f, beta, u, ranges, ...)`

Arguments

For details of shared arguments, see `Emulator`.

`s_diag` The function that modifies the structure of the Bayes Linear adjustment.

`samples` A numeric vector that indicates how many replicates each of the training points has.

`em_type` Whether the emulator is emulating a mean surface or a variance surface.

Constructor Details

See `Emulator`: the constructor structure is the same save for the new arguments discussed above.

Accessor Methods

`get_exp(x, samps = NULL)` Similar in form to the normal Emulator method; the `samps` argument allows the estimation of summary statistics derived from multiple realisations.

`get_cov(x, xp = NULL, full = FALSE, samps = NULL)` Differences here are in line with those described in `get_exp`.

Object Methods

Identical to those of `Emulator`: the one internal difference is that `adjust` returns a HierarchicalEmulator rather than a standard one.

References

Goldstein & Vernon (2016), in preparation

Examples

```
 h_em <- variance_emulator_from_data(BirthDeath$training, c('Y'),
  list(lambda = c(0, 0.08), mu = c(0.04, 0.13)))
 names(h_em) #> c("expectation', 'variance')
```

`idemc` *IDEMC Point Generation*

Description

Performs Implausibility-driven Evolutionary Monte Carlo

Usage

```
idemc(
  ems,
  N,
  targets,
  cutoff = 3,
  s = max(500, ceiling(N/5)),
  sn = s,
  p = 0.4,
  thin = 1,
  pm = 0.9,
  w = 0.8,
  M = 10,
  detailed = FALSE,
  verbose = interactive()
)
```

Arguments

`ems`	The emulators to evaluate implausibility on
`N`	The desired number of points to generate
`targets`	The target values for the outputs
`cutoff`	The desired final implausibility cutoff
`s`	The number of points to generate at each intermediate burn-in stage
`sn`	The number of points to generate in the final burn-in stage
`p`	The proportion of space that should remain in a move along the ladder
`thin`	The thinning factor: a factor T means that N*T points are generated to get N
`pm`	The probability that mutation is chosen in an IDEMC step

`w`	The probability of local random walk moves in the mutation step
`M`	The number of mutations to perform in an IDEMC step, if chosen
`detailed`	If TRUE, points from every ladder rung are returned
`verbose`	Should information about burn-in be displayed during the process?

Details

This method for generating points is focused on finding non-implausible regions that are either extremely small relative to the initial space, are have interesting structure (particularly disconnected structure) that would potentially be overlooked by more standard point generation methods. The method is robust but computationally intensive, and should not be used as a default - for more standard methods of finding points, see `generate_new_runs` and the methods therein.

The IDEMC method operates on an 'implausibility ladder', in the spirit of annealing methods. Each 'rung' of the ladder is characterised by within-cluster and overall variance. The stages performed in one step of the evolutionary algorithm are as follows:

Mutation: A point is modified using a process akin to a random walk step. The parameters that determine the walk can be a global step (determined by the second-order quantities of the entire rung) or a within-cluster step (where the point's cluster is determined and the second-order quantities are drawn from that particular cluster). Local, within-cluster, moves are chosen with probability `w`. The move is retained if the new point satisfies the constraints of its rung.

Crossover: Points are reorganised in descending order of how active their variables are in the emulated outputs, and two different rungs are selected. The points are 'mixed' using a one-point crossover: given a randomly selected index k and two points x1, x2, the new points are y1 = (x11, x12, ..., x1k, x2(k+1), ... x2n) and similarly for y2. The move is retained if both new points satisfy the constraints on their respective rungs. Choices of rung where the first is a later (more restrictive) rung are favoured.

Exchange: Two adjacent rungs are chosen and their points swapped wholesale. The move is retained if the point coming from the less restrictive rung satisfies the constraints of the rung it moves to.

At a given step, only one of mutation or crossover is performed: the probability of performing mutation is given by `pm`. If mutation is chosen, `M` such moves are performed on each rung; if crossover is chosen, then `(n+1)/2` such moves are performed across the `n` rungs. Exchange is always performed and `n+1` such moves are performed.

The choice of 'implausibility ladder' and clusters has a large bearing on the results. This function performs a 'burn-in' to determine a reasonable ladder by starting with a uniform sample across the whole space and defining the next rung by demanding that a percentage (determined by `p`) of the original points satisfy the constraint of this new rung. The IDEMC process is performed on these two rungs to generate `s` points, from which the process is repeated. Once the desired implausibility has been reached, `sn` steps of the algorithm are performed on all rungs to determine final clusters.

Value

Either a list of points (for each rung), or a single data.frame from the last rung.

References

Vernon & Williamson (2013) <arXiv:1309.3520>

See Also

`generate_new_runs` for more standard point generation methods

Examples

```
idemc_res <- idemc(SIREmulators$ems, 500, SIREmulators$targets, s = 250, p = 0.3)
```

`individual_errors` *Predictive Error Plots*

Description

Plots the predictive error with respect to a variety of quantities.

Usage

```
individual_errors(
  em,
  validation,
  errtype = "normal",
  xtype = "index",
  plottype = "normal"
)
```

Arguments

`em`	The emulator to perform diagnostics on
`validation`	The validation set of points with output(s)
`errtype`	The type of individual error to be plotted.
`xtype`	The value to plot against
`plottype`	Whether to plot a standard or Q-Q plot.

Details

The choice of errors to plot is controlled by `errtype`, and can be one of four things: normal, corresponding to the regular standardised errors; eigen, corresponding to the errors after reordering given by the eigendecomposition of the emulator covariance matrix; chol, similarly deriving errors after Cholesky decomposition; and cholpivot, deriving the errors after pivoted Cholesky decomposition.

What the errors are plotted with respect to is controlled by `xtype`. The options are index, which plots them in their order in the validation set; em, which plots errors with respect to the emulator prediction at that point; and any named parameter of the model, which plots with respect to the values of that parameter.

Finally, the plot type is controlled by `plottype`: this can be one of normal, which plots the errors; or qq, which produces a Q-Q plot of the errors.

The default output is to plot the standardised errors (with no decomposition) against the ordering in the validation set; i.e. `errtype = "normal"`, `xtype = "index"`, `plottype = "normal"`.

Some combinations are not permitted, as the output would not be meaningful. Errors arising from an eigendecomposition cannot be plotted against either emulator prediction or a particular parameter (due to the transformation induced by the eigendecomposition); Q-Q plots are not plotted for a non-decomposed set of errors, as the correlation between errors makes it much harder to interpret.

Value

The relevant plot.

References

Bastos & O'Hagan (2009) <doi:10.1198/TECH.2009.08019>

See Also

Other diagnostic functions: `analyze_diagnostic()`, `classification_diag()`, `comparison_diag()`, `get_diagnostic()`, `residual_diag()`, `standard_errors()`, `summary_diag()`, `validation_diagnostics()`

Examples

```
i1 <- individual_errors(SIREmulators$ems$nS, SIRSample$validation)
i2 <- individual_errors(SIREmulators$ems$nS, SIRSample$validation, "chol", "em")
i3 <- individual_errors(SIREmulators$ems$nS, SIRSample$validation, "eigen", plottype = "qq")
i4 <- individual_errors(SIREmulators$ems$nS, SIRSample$validation, "cholpivot", xtype = "aSI")
```

`matern` *Matern correlation function*

Description

For points `x`, `xp`, and a pair of hyperparameters `nu` and `theta`, gives the Matern correlation between the two points.

Usage

```
matern(x, xp, hp)
```

Arguments

`x`	A data.frame of rows corresponding to position vectors
`xp`	A data.frame of rows corresponding to position vectors
`hp`	The hyperparameters nu (smoothness) and theta (correlation length), as a named list

Details

At present, only half-integer arguments for nu are supported.

Value

The Matern correlation between x and xp.

References

Rasmussen & Williams (2005) <ISBN: 9780262182539>

Examples

```
matern(data.frame(a=1), data.frame(a=2), list(nu = 1.5, theta = 0.1))
#> 5.504735e-07
matern(data.frame(a=1,b=2,c=-1),data.frame(a=1.5,b=2.9,c=-0.7), list(nu = 1.5, theta = 0.2))
#> 0.0009527116
```

`nth_implausible` *nth Maximum Implausibility*

Description

Computes the nth-maximum implausibility of points relative to a set of emulators.

Usage

```
nth_implausible(
  ems,
  x,
  z,
  n = 1,
  max_imp = Inf,
  cutoff = NULL,
  sequential = FALSE,
  get_raw = FALSE
)
```

Arguments

`ems`	A set of `Emulator` objects or nested sets thereof (see description)
`x`	An input point, or `data.frame` of points.
`z`	The target values, in the usual form or nested thereof.
`n`	The implausibility level to return.
`max_imp`	A maximum implausibility to return (often used with plotting)
`cutoff`	A numeric value, or vector of such, representing allowed implausibility
`sequential`	Should the emulators be evaluated sequentially?
`get_raw`	Boolean - determines whether nth-implausibility should be applied.

Details

For a collection of emulators, we often combine the implausibility measures for a given set of observations. The maximum implausibility of a point, given a set of univariate emulators and an associated collection of target values, is the largest implausibility of the collected set of implausibilities. The 2nd maximum is the maximum of the set without the largest value, and so on. By default, maximum implausibility will be considered when there are fewer than 10 targets to match to; otherwise second-maximum implausibility is considered.

If `sequential = TRUE` and a specific `cutoff` has been provided, then the emulators' implausibility will be evaluated one emulator at a time. If a point is judged non-implausible by more than `n` emulators, `FALSE` is returned without evaluating any more. Due to R efficiencies, this is more efficient than the 'evaluate all' method once more than around 10 emulators are considered.

This function also deals with variance emulators and bimodal emulators, working in a nested fashion. If targets are provided for both the expectation and variance as a list, then given `ems = list(expectation = ..., variance = ...)` the implausibility is calculated with respect to both sets of emulators, maximising as relevant. If targets are provided in the 'normal' fashion, then only the mean emulators are used. The bimodal case is similar; given a set of emulators `list(mode1 = list(expectation = ..., variance = ...), ...)` then each mode has implausibility evaluated separately. The results from the two modes are combined via piecewise minimisation.

Value

Either the nth maximum implausibilities, or booleans (if cutoff is given).

Examples

```
# A single point
nth_implausible(SIREmulators$ems, data.frame(aSI = 0.4, aIR = 0.25, aSR = 0.025),
 SIREmulators$targets)
# A data.frame of points
grid <- expand.grid(
 aSI = seq(0.1, 0.8, length.out = 4),
 aIR = seq(0, 0.5, length.out = 4),
 aSR = seq(0, 0.05, length.out = 4)
)
# Vector of numerics
i1 <- nth_implausible(SIREmulators$ems, grid, SIREmulators$targets)
# Vector of booleans (same as i1 <= 3)
i2 <- nth_implausible(SIREmulators$ems, grid, SIREmulators$targets, cutoff = 3)
# Throws a warning as n > no. of targets
i3 <- nth_implausible(SIREmulators$ems, grid, SIREmulators$targets, n = 4)
# Vector of booleans (note different output to i2)
i4 <- nth_implausible(SIREmulators$ems, grid, SIREmulators$targets,
 cutoff = c(4, 2.5, 2))

# Variance Emulators
v_ems <- variance_emulator_from_data(BirthDeath$training, c('Y'),
 list(lambda = c(0, 0.08), mu = c(0.04, 0.13)))
v_targs = list(expectation = list(Y = c(90, 110)), variance = list(Y = c(55, 95)))
nth_implausible(v_ems, unique(BirthDeath$validation[,1:2]), v_targs)
## If there is a mismatch between emulators and targets, expectation is assumed
```

```
nth_implausible(v_ems$expectation, unique(BirthDeath$validation[,1:2]), v_targs)
nth_implausible(v_ems, unique(BirthDeath$validation[,1:2]), v_targs$expectation)
```

`orn_uhl` *Ornstein-Uhlenbeck correlation function*

Description

For points `x`, `xp`, and a hyperparameter `theta`, gives the Ornstein-Uhlenbeck correlation between the two points.

Usage

```
orn_uhl(x, xp, hp)
```

Arguments

`x`	A data.frame of rows corresponding to position vectors
`xp`	A data.frame of rows corresponding to position vectors
`hp`	The hyperparameter theta (correlation length) in a named list

Details

This correlation function can be seen as a specific case of the Matern correlation function when nu = 1/2.

Value

The Ornstein-Uhlenbeck correlation between x and xp.

References

Rasmussen & Williams (2005) <ISBN: 9780262182539>

Examples

```
orn_uhl(data.frame(a=1), data.frame(a=2), list(theta = 0.1))
#> 4.539993e-05
orn_uhl(data.frame(a=1,b=2,c=-1),data.frame(a=1.5,b=2.9,c=-0.7), list(theta = 0.2))
#> 0.00469197
orn_uhl(data.frame(a=1,b=1,c=1), data.frame(a=1.2,b=0.9,c=0.6), list(theta = 0.2)) ==
 matern(data.frame(a=1,b=1,c=1), data.frame(a=1.2,b=0.9,c=0.6), list(theta = 0.2, nu = 0.5)) #> TRUE
```

`output_plot` *Emulator Expectation Against Target Outputs*

Description

Plots emulator expectation across the parameter space, with comparison to the corresponding target values (with appropriate uncertainty).

Usage

```
output_plot(ems, targets, points = NULL, npoints = 1000)
```

Arguments

`ems`	The `Emulator` objects.
`targets`	A named list of observations, given in the usual form.
`points`	A list of points at which the emulators should be evaluated.
`npoints`	If no points are provided, the number of input points to evaluate at.

Details

If a `points` data.frame is not provided, then points are sampled uniformly from the input region. Otherwise, the provided points are used: for example, if a representative sample of the current NROY space is available.

Value

A ggplot object

See Also

Other visualisation tools: `behaviour_plot()`, `diagnostic_wrap()`, `effect_strength()`, `emulator_plot()`, `plot_actives()`, `plot_lattice()`, `plot_wrap()`, `simulator_plot()`, `space_removed()`, `validation_pairs()`, `wave_dependencies()`, `wave_points()`, `wave_values()`

Examples

```
output_plot(SIREmulators$ems, SIREmulators$targets)
output_plot(SIREmulators$ems, SIREmulators$targets, points = SIRSample$training)
```

`plot_actives` *Active variable plotting*

Description

For a set of emulators, demonstrate which variables are active.

Usage

```
plot_actives(ems, output_names = NULL, input_names = NULL)
```

Arguments

`ems`	The list of emulators to consider
`output_names`	The names of the outputs to include in the plot, if not all
`input_names`	The names of the inputs to include in the plot, if not all

Details

Each emulator has a list of 'active' variables; those which contribute in an appreciable way to its regression surface. It can be instructive to examine the differences in active variables for a give collection of emulators. The plot here produces an nxp grid for n emulators in p inputs; a square is blacked out if that variable does not contribute to that output.

Both the outputs and inputs can be restricted to collections of interest, if desired, with the optional `output_names` and `input_names` parameters.

Value

A ggplot object corresponding to the plot

See Also

Other visualisation tools: `behaviour_plot()`, `diagnostic_wrap()`, `effect_strength()`, `emulator_plot()`, `output_plot()`, `plot_lattice()`, `plot_wrap()`, `simulator_plot()`, `space_removed()`, `validation_pairs()`, `wave_dependencies()`, `wave_points()`, `wave_values()`

Examples

```
plot_actives(SIREmulators$ems)
# Remove the nR output and aIR input from the plot
plot_actives(SIREmulators$ems, c('nS', 'nI'), c('aSI', 'aSR'))
# Note that we can equally restrict the emulator list...
plot_actives(SIREmulators$ems[c('nS', 'nI')], input_names = c('aSI', 'aSR'))
```

`plot_lattice` *Plot Lattice of Emulator Implausibilities*

Description

Plots a set of projections of the full-dimensional input space.

Usage

```
plot_lattice(ems, targets, ppd = 20, cb = FALSE, cutoff = 3, maxpoints = 50000)
```

Arguments

`ems`	The `Emulator` objects in question.
`targets`	The corresponding target values.
`ppd`	The number of points to sample per dimension.
`cb`	Whether or not a colourblind-friendly plot should be produced.
`cutoff`	The cutoff value for non-implausible points.
`maxpoints`	The limit on the number of points to be evaluated.

Details

The plots are:

One dimensional optical depth plots (diagonal);

Two dimensional optical depth plots (lower triangle);

Two dimensional minimum implausibility plots (upper triangle).

The optical depth is calculated as follows. A set of points is constructed across the full d-dimensional parameter space, and implausibility is calculated at each point. The points are collected into groups based on their placement in a projection to a one- or two-dimensional slice of the parameter space. For each group, the proportion of non-implausible points is calculated, and this value in [0,1] is plotted. The minimum implausibility plots are similar, but with minimum implausibility calculated rather than proportion of non-implausible points.

The `maxpoints` argument is used as a cutoff for if a regular ppd grid would result in a very large number of points. If this is the case, then `maxpoints` points are sampled uniformly from the region instead of regularly spacing them.

Value

A ggplot object

References

Bower, Goldstein & Vernon (2010) <doi:10.1214/10-BA524>

See Also

Other visualisation tools: `behaviour_plot()`, `diagnostic_wrap()`, `effect_strength()`, `emulator_plot()`, `output_plot()`, `plot_actives()`, `plot_wrap()`, `simulator_plot()`, `space_removed()`, `validation_pairs()`, `wave_dependencies()`, `wave_points()`, `wave_values()`

Examples

```
plot_lattice(SIREmulators$ems, SIREmulators$targets, ppd = 10)
plot_lattice(SIREmulators$ems$nS, SIREmulators$targets)
```

`plot_wrap` *Plot proposed points*

Description

A wrapper around R's base plot to show proposed points

Usage

```
plot_wrap(points, ranges = NULL, p_size = 0.5)
```

Arguments

`points`	The points to plot
`ranges`	The parameter ranges
`p_size`	The size of the plotted points (passed to `cex`)

Details

Given a set of points proposed from emulators at a given wave, it's often useful to look at how they are spread and where in parameter space they tend to lie relative to the original ranges of the parameters. This function provides pairs plots of the parameters, with the bounds of the plots calculated with respect to the parameter ranges provided.

Value

The corresponding pairs plot

See Also

Other visualisation tools: `behaviour_plot()`, `diagnostic_wrap()`, `effect_strength()`, `emulator_plot()`, `output_plot()`, `plot_actives()`, `plot_lattice()`, `simulator_plot()`, `space_removed()`, `validation_pairs()`, `wave_dependencies()`, `wave_points()`, `wave_values()`

Examples

```
plot_wrap(SIRSample$training[,1:3], SIREmulators$ems[[1]]$ranges)
```

`problem_data` *Data for an interesting emulation problem*

Description

An RData object consisting of four objects: a data.frame `data` of 208 points, a set `targets` of 19 targets for outputs, a set `ranges` of 21 ranges for inputs, and a data.frame `extra` of 26 additional points. This dataset is used to demonstrate some of the subtleties of emulation in the vignettes, where data transformations can be useful and careful attention should be paid to emulation at early waves.

Usage

```
problem_data
```

Format

A list of objects:

data The training data of 'space-filling' runs

targets The output targets to match to

ranges The input ranges over which the system is valid

extra A set of 'extra' points, generated around a known point of best fit.

`rat_quad` *Rational Quadratic correlation function*

Description

For points `x`, `xp`, and a pair of hyperparameters `alpha` and `theta`, gives the rational quadratic correlation between the two points.

Usage

```
rat_quad(x, xp, hp)
```

Arguments

`x`	A data.frame of rows corresponding to position vectors
`xp`	A data.frame of rows corresponding to position vectors
`hp`	The hyperparameters alpha (exponent and scale) and theta (correlation length)

Details

This correlation function, for d = |x-x'|, has the form $(1 + d^2/(2\alpha\theta^2))^{-\alpha}$, and can be seen as a superposition of exponential-squared correlation functions.

Value

The rational quadratic correlation between x and xp.

References

Rasmussen & Williams (2005) <ISBN: 9780262182539>

Examples

```
rat_quad(data.frame(a=1), data.frame(a=2), list(alpha = 1.5, theta = 0.1))
#> 0.004970797
rat_quad(data.frame(a=1,b=2,c=-1),data.frame(a=1.5,b=2.9,c=-0.7), list(alpha = 1.5, theta = 0.2))
#> 0.02904466
```

`residual_diag` *Emulator Regression Residuals*

Description

Plots the emulator residuals.

Usage

```
residual_diag(emulator, histogram = FALSE, ...)
```

Arguments

`emulator`	The emulator to consider.
`histogram`	Should a histogram or a scatter plot be shown? Default: FALSE
`...`	Any additional arguments (used internally)

Details

An emulator is composed of two parts: a global regression surface, and a local correlation structure. It can sometimes be informative to examine the residuals of the regression surface on the training set, to determine the extent to which the regression surface is being ‘corrected for’ by the correlation structure.

Value

A set of residuals, standardised by the regression surface residual standard error.

See Also

Other diagnostic functions: `analyze_diagnostic()`, `classification_diag()`, `comparison_diag()`, `get_diagnostic()`, `individual_errors()`, `standard_errors()`, `summary_diag()`, `validation_diagnostics()`

Examples

```
residual_diag(SIREmulators$ems$nS)
residual_diag(SIREmulators$ems$nI, TRUE)
```

`simulator_plot` *Plot simulator outputs for multiple waves*

Description

Plots the simulator results for points at successive waves.

Usage

```
simulator_plot(
  wave_points,
  z,
  zero_in = TRUE,
  palette = NULL,
 wave_numbers = seq(ifelse(zero_in, 0, 1), length(wave_points) - ifelse(zero_in, 1,
    0)),
  normalize = FALSE,
  logscale = FALSE,
  barcol = "#444444",
  ...
)
```

Arguments

`wave_points`	The set of wave points, as a list of data.frames
`z`	The set of target values for each output
`zero_in`	Is wave zero included? Default: TRUE
`palette`	If a larger palette is required, it should be supplied here.
`wave_numbers`	Which waves to plot. If not supplied, all waves are plotted.
`normalize`	If true, plotting is done with rescaled target bounds.
`logscale`	If true, targets are log-scaled before plotting.
`barcol`	The colour of the target error bars/bounds
`...`	Optional parameters (not to be used directly)

Details

The values plotted are the outputs from the simulator; the points passed to it are the points suggested by that wave of emulators. By default, wave 0 is included. A colour scheme is chosen outright for all invocations of this function: it is a 10-colour palette. If more waves are required, then an alternative palette should be selected.

Value

A ggplot object.

See Also

Other visualisation tools: `behaviour_plot()`, `diagnostic_wrap()`, `effect_strength()`, `emulator_plot()`, `output_plot()`, `plot_actives()`, `plot_lattice()`, `plot_wrap()`, `space_removed()`, `validation_pairs()`, `wave_dependencies()`, `wave_points()`, `wave_values()`

Examples

```
simulator_plot(SIRMultiWaveData, SIREmulators$targets)
simulator_plot(SIRMultiWaveData[2:4], SIREmulators$targets,
 zero_in = FALSE, wave_numbers = c(1,3))
```

`SIREmulators` *Sample Emulators*

Description

An RData object containing three trained emulators, and the associated targets, for the SIR example. The emulators have been trained on the `SIRSample` training dataset using methods documented in this package.

Usage

```
SIREmulators
```

Format

A list containing two objects:

ems The trained `Emulator` objects.

targets The targets to match to, as a named list.

`SIRImplausibility` *Sample Implausibility Data*

Description

A dataset containing 1000 points from the region bounded by [0.1, 0.8], [0, 0.5], [0, 0.05] for aSI, aIR and aSR respectively. Implausibility has been calculated (for emulators trained on the `SIRSample` training dataset) for each of the outputs nS, nI, nR, and the maximum implausibility is included. The target values used in calculating implausibility were:

nS between 324 and 358

nI mean 143 (sigma 7.15)

nR between 490 and 542

Usage

```
SIRImplausibility
```

Format

A data frame with 1000 rows and 7 variables:

aSI Infection: transition rate from S to I

aIR Recovery: transition rate from I to R

aSR Immunisation: transition rate from S to R

nS Implausibility for nS

nI Implausibility for nI

nR Implausibility for nR

I Maximum implausibility

`SIRMultiWaveData` *Sample Multi-wave Results*

Description

An rda object containing four data.frames: an initial set of points also provided in `SIRSample`, and the 90 points generated at each of three subsequent waves. The trained emulators are provided in `SIRMultiWaveEmulators`.

Usage

```
SIRMultiWaveData
```

Format

A list of data.frame objects:

Wave 0 The initial points used in other examples

Wave 1 Points generated from the wave 1 emulators

Wave 2 Points generated from the wave 2 emulators

Wave 3 Points generated from the wave 3 emulators

`SIRMultiWaveEmulators` *Sample Multi-wave Emulators*

Description

An rda object containing three waves of emulators applied to SIR model (described in `SIRSample`). The corresponding points (both training and validation) are stored in `SIRMultiWaveData`.

Usage

```
SIRMultiWaveEmulators
```

Format

A list containing `Emulator` objects:

Wave 1 Emulators trained on Wave 0, generating wave 1 points

Wave 2 Emulators trained on the results of the above wave 1 points

Wave 3 Emulators trained on the results of the above wave 2 points

---

`SIRSample` *Sample SIR data*

Description

A small dataset containing points generated from a simple deterministic SIR model. The model contains three input parameters, and generates three output parameters. The initial populations are 950 susceptible (S), 50 infected (I), and 0 recovered (R). The final values are taken at time t=10.

Usage

```
SIRSample
```

Format

A list of two data frames. The first has 30 rows and 6 variables, the second 60 rows and 6 variables. The structure is the same in both cases:

aSI Infection: transition rate from S to I

aIR Recovery: transition rate from I to R

aSR Immunisation: transition rate from S to R

nS Final number of S

nI Final number of I

nR Final number of R

Details

The model operates using simple differential equations, where

S’ = aSR*R - aSI*S*R/(S+I+R)

I’ = aSI*S*R/(S+I+R) - aIR*I

R’ = aIR*I - aSR*R.

`SIR_stochastic` *Stochastic SIR Data*

Description

An RData object consisting of two data.frames (in a similar fashion to `BirthDeath`). The first consists of 30 points in the parameter space (aSI, aIR, aSR), each of which has been inputted into the Gillespie algorithm for the stochastic version of the model used in `GillespieSIR` (but with changed starting conditions) 100 times. The second has similar form but for 20 unique points, each with 100 repetitions.

Usage

```
SIR_stochastic
```

Format

A list of two data.frames `training` and `validation`: each has the following columns:

aSI Infection rate

aIR Recovery rate

aSR Waning immunity rate

I10 (25, 50) The number of infected people at t = 10 (25, 50)

R10 (25, 50) The number of recovered people at t = 10 (25, 50)

Details

The outputs observed are the numbers of infected (I) and recovered (R) people at time points t = 10, 25, 50. All outputs display some level of bimodality. The initial conditions to generate the runs had S(0)=995, I(0)=5, R(0)=0.

`space_removal` *Percentage of Space Removed*

Description

For a wave of emulators, estimates the proportion of space removed at this wave.

Usage

```
space_removal(ems, targets, points = NULL, cutoff = 3, individual = TRUE)
```

Arguments

`ems`	The emulators to compute over, as a list
`targets`	The output target values
`points`	The points to test against
`cutoff`	The cutoff value for implausibility
`individual`	If true, gives emulator-by-emulator results; otherwise works with maximum implausibility

Details

Given a collection of emulators corresponding to a wave, we can look at an estimate of the proportion of points from previous waves that will be accepted at this wave, either on an emulator-by-emulator basis (to see which outputs are most restrictive) or as an all-wave determination.

Naturally, such a statement will be an estimate of the restriction on the full space (which will become more unreliable for higher dimensions), but it can give an order-of-magnitude statement, or useful comparators between different emulators in a wave.

If no points are provided, the training points for the emulators are used. For best results, a good number of points should be given: typically one should consider using as many points as one knows to be in the NROY space (including any validation points, if accessible).

Value

A numeric corresponding to the proportions of points accepted.

See Also

`space_removed` for a visualisation of the space removal.

Examples

```
space_removal(SIREmulators$ems, SIREmulators$targets,
 rbind(SIRSample$training, SIRSample$validation))
space_removal(SIREmulators$ems, SIREmulators$targets,
 rbind(SIRSample$training, SIRSample$validation), individual = FALSE)
```

`space_removed` *Space Removal Diagnostics*

Description

Finds the proportion of space removed as a function of implausibility cut-off and of one of structural discrepancy, emulator variance, or correlation hyperparameter(s).

Usage

```
space_removed(
  ems,
  targets,
  ppd = 10,
  u_mod = seq(0.8, 1.2, by = 0.1),
  intervals = seq(0, 10, length.out = 200),
  modified = "obs",
  maxpoints = 50000
)
```

Arguments

`ems`	The `Emulator` objects.
`targets`	The corresponding targets to match to.
`ppd`	The number of points per input dimension to sample at.
`u_mod`	The proportional values by which to inflate/deflate the relevant statistic.
`intervals`	The interval values of the implausibility cutoff at which to evaluate.
`modified`	The statistic to modify: obs, disc, var or hp (see above)
`maxpoints`	The maximum number of points to evaluate at

Details

The reduction in space is found by evaluating a p^d regular grid, where p is chosen by ppd and d is the dimension of the input space. Larger values of p will give a more accurate reflection of the space removed, at a corresponding computational cost. For the purpose of quick-and-dirty diagnostics, ppd = 5 is sufficient: the default is 10.

The parameter `modified` can be one of three strings: `'obs'` corresponding to observation uncertainty; `'disc'` corresponding to internal and external discrepancy (as given in `Emulator$disc`);

`'var'` corresponding to global emulator variance (as given by `Emulator$u_sigma`), and `'hp'` corresponding to the hyperparameters of the emulator correlation structure. In the first case, the implausibilities are recalculated for each inflation value; in the other two cases the emulators are retrained. For this reason, the `'var'` and `'hp'` options are computationally more intensive. The default is `'obs'`.

The inflationary/deflationary values are chosen by `u_mod`: the default is to take 80%, 90%, 100%, 110%, and 120% of the original value as the variation. The proportion of points deemed non-implausible is checked at a set of implausibility cutoffs defined by `intervals`, and a plot is returned showing the relevant data.

Value

A ggplot object

See Also

`space_removal` for a numeric representation of space removed.

Other visualisation tools: `behaviour_plot()`, `diagnostic_wrap()`, `effect_strength()`, `emulator_plot()`, `output_plot()`, `plot_actives()`, `plot_lattice()`, `plot_wrap()`, `simulator_plot()`, `validation_pairs()`, `wave_dependencies()`, `wave_points()`, `wave_values()`

Examples

```
space_removed(SIREmulators$ems, SIREmulators$targets, ppd = 5)
space_removed(SIREmulators$ems$nS, SIREmulators$targets,
 ppd = 5, u_mod = seq(0.75, 1.25, by = 0.25), intervals = seq(2, 6, by = 0.1))
```

`standard_errors` *Standardized Error Diagnostics*

Description

Shorthand function for diagnostic test ‘se’.

Usage

```
standard_errors(
  emulator,
  targets = NULL,
  validation = NULL,
  plt = interactive()
)
```

Arguments

`emulator`	The emulator in question
`targets`	The output targets
`validation`	The validation set
`plt`	Whether to plot or not

Details

For details of the function, see `get_diagnostic` and for the plot see `analyze_diagnostic`.

Value

A data.frame of failed points

References

Jackson (2018) <http://etheses.dur.ac.uk/12826>

See Also

`get_diagnostic`, `analyze_diagnostic`, `validation_diagnostics`

Other diagnostic functions: `analyze_diagnostic()`, `classification_diag()`, `comparison_diag()`, `get_diagnostic()`, `individual_errors()`, `residual_diag()`, `summary_diag()`, `validation_diagnostics()`

`subset_emulators` *Subsetting for Bimodal/Variance Emulators*

Description

Takes a collection of bimodal or stochastic emulators and subsets by output name.

Usage

```
subset_emulators(emulators, output_names)
```

Arguments

`emulators`	A set of emulators, often in nested form
`output_names`	The names of the desired outputs

Details

It can be useful to consider only a subset of outputs. In the normal case, this can be easily achieved; however, when the emulators are in a nested structure such as that provided by variance_emulator_from_data or bimodal_emulator_from_data, it can be more involved. This function allows the easy selecting of emulators by name, returning a subset of them in the same form as the original object.

This function is compatible with 'standard' emulators; that is, those in a simple list, equivalent to subsetting over the collection of output names of the emulators that exist in `output_names`.

Value

An object of the same form as 'emulators'.

`summary_diag` *Summary Statistics for Emulators*

Description

Generates measures for emulator quality

Usage

```
summary_diag(emulator, validation, verbose = interactive())
```

Arguments

`emulator`	The emulator to test
`validation`	The validation set, consisting of points and output(s)
`verbose`	Should statistics be printed out?

Details

A couple of summary statistics can be generated for emulators, based on their prediction errors on a validation set. This function produces the test statistic for a comparison to a relevant chi-squared distribution, and the similar test statistic for an F-distribution. In both cases, the expectation and standard deviation of the underlying distribution are also provided.

The output of this function is a logical vector stating whether the derived value lies within 3-sigma of the expected value. In systems where errors are expected to be correlated, higher weight should be given to the Mahalanobis measure than the chi-squared measure. Any anomalous results can be investigated in more depth using the `individual_errors` function.

Value

Whether the observed value lies within 3-sigma of the expected value.

References

Bastos & O'Hagan (2009) <doi:10.1198/TECH.2009.08019>

See Also

Other diagnostic functions: `analyze_diagnostic()`, `classification_diag()`, `comparison_diag()`, `get_diagnostic()`, `individual_errors()`, `residual_diag()`, `standard_errors()`, `validation_diagnostics()`

Examples

```
summary_diag(SIREmulators$ems$nR, SIRSample$validation)
```

---

validation_diagnostics

Emulator Diagnostics

Description

Performs the standard set of validation diagnostics on emulators.

Usage

```
validation_diagnostics(
  emulators,
  targets = NULL,
  validation = NULL,
  which_diag = c("cd", "ce", "se"),
  analyze = TRUE,
  diagnose = "expectation",
  ...
)
```

Arguments

`emulators`	A list of `Emulator` objects.
`targets`	The list of observations for the outputs
`validation`	The validation set, containing all inputs and outputs.
`which_diag`	Which diagnostics should be performed (see description)
`analyze`	Should plotting and/or failing points be returned?
`diagnose`	For bimodal systems, should the expectation or variance be considered?
`...`	Any additional parameters to pass to the diagnostics (eg sd, cutoff, ...)

Details

All the diagnostics here can be performed with or without a validation (or ‘holdout’) set of data. The presence of a set of targets is optional for some checks but mandatory for others: the appropriate warnings will be given in the event that some checks cannot be applied.

The current options for diagnostics (with the codes for `which_diag`) are:

Standardised Errors (se)

Comparison Diagnostics (cd)

Classification Errors (ce)

All of the above (all)

For details of each of the tests, see the help file for `get_diagnostic`.

Value

A data.frame containing points that failed one or more diagnostic tests.

See Also

Other diagnostic functions: `analyze_diagnostic()`, `classification_diag()`, `comparison_diag()`, `get_diagnostic()`, `individual_errors()`, `residual_diag()`, `standard_errors()`, `summary_diag()`

Examples

```
validation_diagnostics(SIREmulators$ems, SIREmulators$targets, SIRSample$validation)
# data.frame of failed points (empty) and a 3x3 set of plots
validation_diagnostics(SIREmulators$ems, SIREmulators$targets, SIRSample$validation,
 c('ce','cd'))
# empty data.frame and a 3x2 set of plots
validation_diagnostics(SIREmulators$ems, SIREmulators$targets, SIRSample$validation,
 cutoff = 2, sd = 2)
# k-fold (with k = 3)
validation_diagnostics(SIREmulators$ems, SIREmulators$targets, k = 3)
```

`validation_pairs` *Validation Set Diagnostics and Implausibility*

Description

Creates pairs plots on the set of validation points of diagnostic suitability and implausibility.

Usage

```
validation_pairs(ems, points, targets, ranges, nth = 1, cb = FALSE)
```

Arguments

`ems`	The `Emulator` object(s).
`points`	The set of validation points to plot.
`targets`	The set of targets to match to.
`ranges`	If provided, this gives the plotting region (see above).
`nth`	The level of maximum implausibility to plot.
`cb`	Whether or not the colour scheme should be colourblind friendly.

Details

The plots are organised as follows:

a) Emulated versus simulated output (lower diagonal). This is similar in spirit to `comparison_diag`: the plotted points are their location in the input space and the points are coloured by the emulator prediction's deviation from the simulator value.

b) Implausibility (upper diagonal). The points are again plotted based on their location in input space, but their colouration is now based on the implausibility of the point.

If `ranges` is provided, then the plotting region is created relative to these ranges. This can be useful if on later waves of a history match and the plotting is to be done relative to the original input space, rather than the (reduced) parameter space upon which the emulators have been trained.

Value

A ggplot object.

See Also

Other visualisation tools: `behaviour_plot()`, `diagnostic_wrap()`, `effect_strength()`, `emulator_plot()`, `output_plot()`, `plot_actives()`, `plot_lattice()`, `plot_wrap()`, `simulator_plot()`, `space_removed()`, `wave_dependencies()`, `wave_points()`, `wave_values()`

Examples

```
validation_pairs(SIREmulators$ems, SIRSample$validation, SIREmulators$targets)
wider_ranges <- purrr::map(SIREmulators$ems[[1]]$ranges, ~.*c(-2, 2))
validation_pairs(SIREmulators$ems, SIRSample$validation,
 SIREmulators$targets, ranges = wider_ranges, cb = TRUE)
```

`variance_emulator_from_data`

Variance Emulator Creation

Description

Trains hierarchical emulators to stochastic systems

Usage

```
variance_emulator_from_data(
  input_data,
  output_names,
  ranges,
  input_names = names(ranges),
  verbose = interactive(),
  ...
)
```

Arguments

`input_data`	All model runs at all points.
`output_names`	The observation names.
`ranges`	A named list of parameter ranges
`input_names`	The names of the parameters (if `ranges` is not provided).
`verbose`	Should status updates be printed to console?
`...`	Optional parameters that can be passed to `link{emulator_from_data}`.

Details

For stochastic systems, one may emulate the variance as well as the function itself. This is particularly true if one expects the variance to be very different in different areas of the parameter space (for example, in an epidemic model). This function performs the requisite two-stage Bayes Linear update.

All observations are required (including replicates at points) - this function collects them into the required chunks and calculates the summary statistics as required.

All other parameters passed to this function are equivalent to those in emulators are the Bayes Linear adjusted forms.

Value

A list of lists: one for the variance emulators and one for the function emulators.

References

Goldstein & Vernon (2016) in preparation

Examples

```
# A simple example using the BirthDeath dataset
v_ems <- variance_emulator_from_data(BirthDeath$training, c("Y"),
 list(lambda = c(0, 0.08), mu = c(0.04, 0.13)), c_lengths = c(0.75))
```

`wave_dependencies` *Multiple Wave Inputs vs Outputs*

Description

Given multiple waves of points, produce input-output plots for each pair.

Usage

```
wave_dependencies(
  waves,
  targets,
  output_names = names(targets),
  input_names = names(waves[[1]])[!names(waves[[1]]) %in% names(targets)],
  p_size = 1.5,
  l_wid = 1.5,
  normalize = FALSE,
  zero_in = TRUE,
  wave_numbers = ifelse(zero_in, 0, 1):(length(waves) - ifelse(zero_in, 1, 0)),
  ...
)
```

Arguments

`waves`	The list of data.frame objects, one for each set of outputs at that wave.
`targets`	The target values of the outputs.
`output_names`	The outputs to plot, if not all are wanted.
`input_names`	The inputs to plot, if not all are wanted.
`p_size`	Control for the point size on the plots: smaller is better for many plots.
`l_wid`	Control for line width of superimposed targets.
`normalize`	If true, plotting is done with target bounds equal size.
`zero_in`	Is a wave 0 included in the waves list?
`wave_numbers`	Which waves to plot
`...`	Optional parameters (not to be used directly)

Details

It can be useful to consider what the dependencies between the input values and output values are, to investigate the suitability of the chosen input ranges (i.e. if widening an input range could result in the targets being matchable). This function provides those plots.

For each output-input pair, a points plot is produced with the input value on the x-axis and the output value on the y-axis. The target bounds are superimposed as horizontal lines. The points themselves are coloured by which wave of history matching they came from.

These can show dependencies between specific outputs and inputs and, if points are clustering at the far left or right edge of a plot, can give an indication that the input ranges are unsuitable for matching the target.

Value

A grid of ggplot objects.

See Also

Other visualisation tools: `behaviour_plot()`, `diagnostic_wrap()`, `effect_strength()`, `emulator_plot()`, `output_plot()`, `plot_actives()`, `plot_lattice()`, `plot_wrap()`, `simulator_plot()`, `space_removed()`, `validation_pairs()`, `wave_points()`, `wave_values()`

Examples

```
 wave_dependencies(SIRMultiWaveData, SIREmulators$targets, l_wid = 0.8, p_size = 0.8)
wave_dependencies(SIRMultiWaveData, SIREmulators$targets, c('nS', 'nI'), c('aIR', 'aSI'))

    # For many plots, it may be helpful to manually modify the font size
    wave_dependencies(SIRMultiWaveData, SIREmulators$targets) +
     ggplot2::theme(text = ggplot2::element_text(size = 5))
```

`wave_points` *Multiple Wave Point Plotting*

Description

Given multiple waves of points, produces pairs plots

Usage

```
wave_points(
  waves,
  input_names,
  surround = FALSE,
  p_size = 1.5,
  zero_in = TRUE,
  wave_numbers = ifelse(zero_in, 0, 1):(length(waves) - ifelse(zero_in, 1, 0)),
  ...
)
```

Arguments

`waves`	The list of data.frames, one for each set of points at that wave.
`input_names`	The input names to be plotted.
`surround`	If true, points are surrounded by black boundaries.
`p_size`	The size of the points. Smaller values are better for high-dimensional spaces.
`zero_in`	Is a wave 0 included in the waves list?
`wave_numbers`	Which waves to plot
`...`	Optional parameters (not to be used directly)

Details

Subsequent waves are overlaid on the same pairs plots, to determine the evolution of the non-implausible region. One-dimensional density plots are also created on the diagonal.

Value

A ggplot object

See Also

Other visualisation tools: `behaviour_plot()`, `diagnostic_wrap()`, `effect_strength()`, `emulator_plot()`, `output_plot()`, `plot_actives()`, `plot_lattice()`, `plot_wrap()`, `simulator_plot()`, `space_removed()`, `validation_pairs()`, `wave_dependencies()`, `wave_values()`

Examples

```
 wave_points(SIRMultiWaveData, c('aSI', 'aIR', 'aSR'))

     wave_points(SIRMultiWaveData, c('aSI', 'aIR', 'aSR'), TRUE, 0.8)
     # For many plots, it may be helpful to manually modify the font size
     wave_points(SIRMultiWaveData, c('aSI', 'aIR', 'aSR')) +
      ggplot2::theme(text = ggplot2::element_text(size = 5))
```

`wave_values` *Multiple Wave Output Plotting*

Description

Given multiple waves of points, produces pairs plots of the outputs.

Usage

```
wave_values(
  waves,
  targets,
  output_names = names(targets),
  ems = NULL,
  surround = FALSE,
  restrict = FALSE,
  p_size = 1.5,
  l_wid = 1.5,
  zero_in = TRUE,
  wave_numbers = ifelse(zero_in, 0, 1):(length(waves) - ifelse(zero_in, 0, 1)),
  which_wave = ifelse(zero_in, 0, 1),
  upper_scale = 1,
  ...
)
```

Arguments

`waves`	The list of data.frames, one for each set of outputs at that wave.
`targets`	The output targets.
`output_names`	The outputs to plot.
`ems`	If provided, plots the emulator expectations and 3-standard deviations.
`surround`	As in `wave_points`.
`restrict`	Should the plotting automatically restrict to failing target windows?
`p_size`	As in `wave_points`.
`l_wid`	The width of the lines that create the target boxes.
`zero_in`	Is a wave 0 included in the waves list?
`wave_numbers`	Which waves to plot.
`which_wave`	Scaling for lower plots (see description)
`upper_scale`	Scaling for upper plots (ibid)
`...`	Optional parameters (not to be used directly)

Details

This function operates in a similar fashion to `wave_points` - the main difference is that the output values are plotted. Consequently, the set of targets is required to overlay the region of interest onto the plot.

To ensure that the wave numbers provided in the legend match, one should provide waves as a list of data.frames with the earliest wave at the start of the list.

The parameters `which_wave` and `upper_scale` control the level of ‘zoom’ on each of the lower-triangular and upper-triangular plots, respectively. For the lower plots, `which_wave` determines which of the provided waves is to be used to determine the output ranges to plot with respect to: generally, higher `which_wave` values result in a more zoomed-in plot. For the upper plots, `upper_scale` determines the plot window via a multiple of the target bounds: higher values result in a more zoomed-out plot. If not provided, these default to `which_wave=0` (or 1 if no wave 0 is given) and `upper_scale = 1`. If the value provided to `which_wave` does not correspond to a provided wave (or one explicitly not included in `wave_numbers`), it defaults to the closest available wave to the value of `which_wave`.

If `ems` is provided, it should follow the same structure as `waves`: at the very least, it should contain all emulators trained over the course of the waves. The emulator predictions for a target are made by the emulator for that target whose ranges are the smallest such that contain the point.

Value

A ggplot object.

See Also

Other visualisation tools: `behaviour_plot()`, `diagnostic_wrap()`, `effect_strength()`, `emulator_plot()`, `output_plot()`, `plot_actives()`, `plot_lattice()`, `plot_wrap()`, `simulator_plot()`, `space_removed()`, `validation_pairs()`, `wave_dependencies()`, `wave_points()`

Examples

```
wave_values(SIRMultiWaveData, SIREmulators$targets, surround = TRUE, p_size = 1)

  wave_values(SIRMultiWaveData, SIREmulators$targets, c('nS', 'nI'), l_wid = 0.8)
    wave_values(SIRMultiWaveData, SIREmulators$targets, l_wid = 0.8,
     wave_numbers = c(0, 1, 3), which_wave = 2, upper_scale =  1.5)
    # For many plots, it may be helpful to manually modify the font size
    wave_values(SIRMultiWaveData, SIREmulators$targets) +
     ggplot2::theme(text = ggplot2::element_text(size = 5))
```

Index