Package 'emulatorr'

July 13, 2020

Type Package

Title Emulation and History Matching Package

Version 0.4.0

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

Description
A set of functions for Bayes Linear emulation and history matching. For details on the mathematical background, there are many papers freely available on the topic; for details of the functions in this package, consult the manual or the help files for individual functions or objects.

Suggests knitr, rmarkdown

VignetteBuilder knitr

Depends R (>= 3.5.0)

Imports lhs,
purrr,
R6,
ggplot2,
viridis,
viridisLite,
reshape2,
plyr,
patchwork,
cowplot,
mvtnorm,
GGally,
rlang

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0

R topics documented:

`classification_error` *Classification diagnostics*

Description

Checks for emulator misclassifications. Both the emulator implausibility and the simulator implausibility are computed, and plotted against one another. Points for which the emulator implausibility is outside the desired cut-off but for which the simulator implausibility is not are misclassification points, and are highlighted in red.

Usage

```
classification_error(
  emulator,
  input_points,
  output_points,
  z,
  output_name,
  cutoff = 3,
  plt = T,
  ...
)
```

Arguments

emulator	An Emulator object.
input_points	A set of validation points.
output_points	The outputs, f(x), from the simulator.
z	The observation to test implausibility against. Either as a single numeric, or as list(val=numeric,sigma=numeric).
output_name	Optional. A name for the output.
cutoff	Optional. The cut-off for the implausibility measure.
plt	Should a plot be shown (default: T).
...	Dummy parameters (for compatibility with diagnostic wrapper)

Value

The set of points misclassified by the emulator.

Examples

```
em <- emulator_from_data(GillespieSIR, output_names = c('nS', 'nI', 'nR'),
 ranges = list(aSI = c(0.1, 0.8), aIR = c(0, 0.5), aSR = c(0, 0.05)),
 quadratic = TRUE)[[1]]
target_value <- list(val = 281, sigma = 37.26)
classification_error(em, GillespieValidation[,1:3], GillespieValidation[,'nS'], target_value)
#> data.frame containing 0 points
```

---

comparison_diagnostics

Emulator Diagnostic Plot

Description

Produces a diagnostic plot of emulator output. The emulator output E[f(x)] is plotted against the simulator output f(x), with error bars given by the emulator standard deviation sqrt(3*Var[f(x)]), for each point x in a validation set X. Points whose emulator expectation lies outside 3-sigma of the simulator output are shown in red, and those input points are returned.

Usage

```
comparison_diagnostics(
  emulator,
  input_points,
  output_points,
  output_name,
  sd = 3,
  plt = T,
  ...
)
```

Arguments

emulator	An Emulator object.
input_points	A set of validation points.
output_points	The outputs, f(x), from the simulator.
output_name	Optional. A name for the output.
sd	Numeric: the allowed number of standard deviations (default: 3).
plt	Should a plot be shown (default: T).
...	Dummy parameters (for compatibility with diagnostic wrapper)

Value

A list of points whose emulator value is outside the allowed standard deviation.

Examples

```
em <- emulator_from_data(GillespieSIR, output_names = c('nS', 'nI', 'nR'),
 ranges = list(aSI = c(0.1, 0.8), aIR = c(0, 0.5), aSR = c(0, 0.05)),
 quadratic = TRUE)[[1]]
comparison_diagnostics(em, GillespieValidation[,1:3], GillespieValidation[,'nS'])
#> (0.7864384, 0.01426296, 0.001072935)
```

Emulator *Bayes Linear Emulator*

Description

Creates an 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,bucov = NULL,data = NULL,delta = 0)
```

Arguments

basis_f A list of basis functions to be used. For ease of understanding, it is advisable to arrange these in increasing powers of the variables. The constant function function(x) 1 should be the first element (this is the format given if emulator_from_data is used to generate emulators).

beta A set of regression parameters. These are provided in the form list(mu,sigma), where mu are the expectations of the coefficients and sigma the corresponding covariance matrix.

u The specifications for the correlation structure. This has three parts: the expectation E[u(x)], the variance Var[u(x)], and a correlation function c(x,x'). These are passed as list(mu,sigma,corr).

ranges A named list of ranges for the input parameters. Required: if, for example, we have two inputs a and b with ranges [-0.5,0.5] and [3,5] respectively, then ranges = list(a = c(-0.5,0.5),b = (3,5)).

bucov The covariance between the regression parameters and the correlation structure, as a vector of length `length(beta$mu)`. Preferably this should be defined as a list of functions.

`data` If an adjusted emulator is desired, then the data by which to adjust is specified here, as a `data.frame` with named columns.

`delta` A nugget to add to the correlation structure, in the range [0,1).

Constructor Details

The constructor must take a list of vectorised basis functions, whose length is equal to the number of regression coefficients, or an error will be thrown. The correlation structure should be stationary, or at least such that we can define `sigma` as a global variance: if an adjusted emulator is required, we supply an unadjusted u(x) and the corresponding data by which to adjust. The Bayes Linear update equations will then provide the modified (generally non-stationary) correlation structure. This has the advantage that if, after diagnostics, we need to inflate or deflate the overal variance, we can simply modify `sigma`.

The use of a nugget is as follows. If we have a generic correlation structure with Cov[u(x), u(x')] = sigma^2*c(x,x'), where c(x,x') is some correlation function, then the addition of a nugget transforms this to Cov[u(x), u(x')] = sigma^2*(1-delta)*c(x,x')+sigma^2*delta*I(x,x'), where I(x,x') is an indicator function. The nugget maintains the variance at a point while deflating the covariance between points.

Accessor Methods

`get_exp(x)` Returns the emulator expectation at a collection of points, `x`.

`get_cov(x,xp=NULL,full = FALSE)` Returns the covariance between collections of points `x` and `xp`. If no `xp` is supplied, then the covariance matrix for the points in `x` is calculated; if `full = TRUE`, the full covariance matrix is calculated; otherwise only the variances Var[f(x), f(x')] are calculated for each x in `x` and x' in `xp`.

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

`implausibility(x,z)` Returns the implausibility that input points `x` could give rise to an output `z`. The output `z` can be supplied in two ways: either as a list `z = list(val,sigma)` where `val` is the output and `sigma` the corresponding uncertainty (e.g. observation error, model discrepancy); or as a single numeric. In the latter case, the uncertainty is assumed to be identically 0 (tread with caution in these circumstances!)

Object Methods

`adjust(data,out_name)` Performs Bayes Linear adjustment, given the data. The data should contain all input parameters (even if they are not necessarily active for this emulator) and the single output. This function creates a new emulator object with the adjusted expectation and variance of beta as the primitive specifications, and supplies the data for the new emulator to compute the adjusted expectation and variance of u(x), and the adjusted Cov[beta, u(x)].

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 = function(x, xp) exp_sq(x, xp, 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
em$get_exp(c(0.1, 0.1)) #> 0.6
em$get_cov(c(0.1, 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, sample_points_2, full = TRUE) # Returns a 16x12 matrix of covariances

b_u_cov <- function(x) c(1, x[[1]], x[[1]]*x[[2]])
del <- 0.1
all_specs_em <- Emulator$new(basis_functions, beta, u, ranges,
 bucov = b_u_cov, delta = del)
all_specs_em$get_exp(c(0.1, 0.1)) #> 0.6
all_specs_em$get_cov(c(0.1, 0.1)) #> 11.60844

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

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
```

`emulator_from_data` *Generate Prior Emulators from Data*

Description

Given data from a simulation, generates a set of `Emulator` objects based on fitted values.

Usage

```
emulator_from_data(
  input_data,
  output_names,
  ranges,
  input_names = names(ranges),
  beta,
  u,
  c_lengths,
  funcs,
  bucov,
  deltas,
  quadratic = F,
  beta.var = F
)
```

Arguments

input_data	Required. A data.frame containing the input parameters and output values from a set of simulator runs.
output_names	Required. The list of outputs to emulate from input_data.
ranges	A named list of parameter ranges.
input_names	A list of input_names (if ranges is not provided).
beta	Optional: specifications for the regression coefficients, given as a list of lists list(mu,sigma) (a la Emulator specification).
u	Optional: the correlation structure for each output, given as a list of lists list(mu,sigma,corr).
c_lengths	Optional: a set of correlation lengths.
funcs	Optional: basis functions for the regression surface.
bucov	Optional: a list of functions giving the covariance between each of the beta parameters and u(x).
deltas	Optional: the nugget terms to include in u(x).
quadratic	Optional: should the regression surface be linear or quadratic? Default: F
beta.var	Optional: should the beta coefficient be assumed to be known or should model variance be included?

Details

Many of the parameters that can be passed to this function are optional; the bare minimum is input_data, output_names, and one of ranges or input_names. If ranges is specified, then the input names are taken from that; if only input_names is specified, then it is assumed that all input values in input_data are already scaled to [-1, 1].

If the minimum information is provided, then a model is fitted as follows.

The basis functons and regression coefficients are generated using the lm function using either only linear terms or up to quadratic terms (dependent on the value of quadratic), performing stepwise add or delete as appropriate; in either event, the AIC criteria is used to select the terms. The regression parameters thus derived are assumed to be known if beta.var=FALSE, so that beta$sigma = diag(0). Otherwise, the covariance matrix for the parameters is taken from vcov(model).

The correlation function c(x,x') is taken to be exp_sq; the correlation length is chosen using the Durham heuristic: this states that the correlation length should lie within [1/(n+1), 2/(n+1)] where n is the degree of the fitted surface (and the range of the parameter is [-1,1]). A value in this range is sampled uniformly: it may be useful to generate emulators allowing this randomness and use diagnostics to decide on a fixed value. The nugget is assumed to be 0 unless otherwise specified. The expectation E[u(x)] is assumed to be 0, and the variance sigma^2 is taken from the residual squared error from the model used to fit the basis functions and betas.

The covariance between beta and u(x) is assumed to vanish.

Value

A list of objects of class Emulator.

Examples

```
# Use the GillespieSIR dataset
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(GillespieSIR, output_names = out_vars,
 ranges = ranges)
ems_linear # Printout of the key information

ems <- emulator_from_data(GillespieSIR, output_names = out_vars,
 ranges = ranges, quadratic = TRUE)
ems # Now includes quadratic terms (but only where they're warranted)

ems2 <- emulator_from_data(GillespieSIR, output_names = out_vars,
 ranges = ranges, c_lengths = c(0.55, 0.6, 0.59),
 deltas = c(0.1, 0.2, 0.2), quadratic = TRUE)
ems2 # Broadly the same, but with the correlation structure modified.

ems2_beta <- emulator_from_data(GillespieSIR, output_names = out_vars,
 ranges = ranges, c_lengths = c(0.55, 0.6, 0.59),
 deltas = c(0.1, 0.2, 0.2), quadratic = TRUE, beta.var = TRUE)
```

`emulator_plot` *Plot Emulator Outputs*

Description

A wrapper for plotting emulator outputs, for two dimensions. If the input space is greater than 2-dimensional, the mid-range values are chosen for any input beyond the first two. If a list of k emulators are given (e.g. those derived from `emulator_from_data` or `full_wave`), then the result is a kx2 grid of plots. If a single emulator is given, then a single plot is returned.

Usage

```
emulator_plot(
  em,
  var_name = "exp",
  npoints = 40,
  targets = NULL,
  cb = FALSE,
  ...
)
```

Arguments

`em`	A single `Emulator` object, or a list thereof
`var_name`	The output to plot: options are described above
`npoints`	The number of lattice points per input direction
`targets`	The observations. Required if plotting implausibility
`cb`	Should implausibility plots be coloured as colourblind friendly?
`...`	Any optional parameters for `nth_implausible`

Details

Options for plotting variables are passed via the `var` parameter: current choices are `'exp'` (Expectation), `'var'` (Variance), `'imp'` (Implausibility), and `'maximp'` (n-th maximum Implausibility). If either of the implausibilities are desired, the `targets` parameter must not be `NULL`. Bear in mind that n-th maximum implausibility is only permitted if a list of emulators is provided: the `...` parameters that can be passed to this function are for optional parameters that can be passed to the `nth_implausible` function (for example, which level of maximum implausibility is wanted).

Value

A ggplot object.

Examples

```
ranges <- list(aSI = c(0.1, 0.8), aIR = c(0, 0.5), aSR = c(0, 0.05))
targets <- list(
 list(val = 281, sigma = 10.43),
 list(val = 30, sigma = 11.16),
 list(val = 689, sigma = 14.32)
)
outputs <- c('nS','nI','nR')
ems <- emulator_from_data(GillespieSIR, outputs, ranges, deltas=rep(0.1, 3), quadratic = TRUE)
t_ems <- purrr::map(seq_along(ems), ~ems[[.]]$adjust(GillespieSIR, outputs[[.]]))
names(t_ems) <- outputs
emulator_plot(t_ems$nI)
emulator_plot(t_ems, var_name = 'var', npoints = 10)
emulator_plot(t_ems, var_name = 'sd', npoints = 10)
emulator_plot(t_ems, var_name = 'imp', targets = targets, npoints = 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, theta)
```

Arguments

`x`	A numeric position vector
`xp`	A numeric position vector
`theta`	A numeric correlation length

Value

The exponental-squared correlation between x and xp.

Examples

```
exp_sq(1,2,0.1)
#> 3.720076e-44
exp_sq(c(1,2,-1),c(1.5,2.9,-0.7),0.2)
#> 3.266131e-13
```

`full_wave` *History Match*

Description

Performs a full wave of emulation and history matching from data. Given simulator runs (split into training and validation data), the target values, and the identification of outputs to emulate, the function generates trained emulators, tests them with emulator diagnostics (removing any emulators whose outputs cannot be well emulated from the data), and finally generates a new sample of points to be entered into the simulator.

Usage

```
full_wave(
  input_data,
  validation_data,
  ranges,
  output_names,
  targets,
  n_points = 40,
  previous_wave = NULL,
  sample_method = "importance",
  ...
)
```

Arguments

`input_data`	The set of training points
`validation_data`	The set of points to use in validation
`ranges`	The ranges of the inputs, as a named list.
`output_names`	The names of the outputs to emulate.
`targets`	The observations, given in the usual `(val,sigma)` form
`n_points`	The number of points to evaluate the parameters on.
`previous_wave`	The preliminary emulators for the set of waves, if they exist
`sample_method`	The method to be used to find new points (see `generate_new_runs`)
`...`	Any optional parameters to pass to `emulator_from_data`

Details

Necessary parameters to be passed are the input data, the validation data, the ranges of inputs, and the observation values for each output. If any specifications are to be passed directly to the emulator construction, then they should be given as additional parameters (see `emulator_from_data` to see the options).

If the wave to be emulated is the first wave, then a set of preliminary ('wave 0') emulators are fitted to the data. The proper set of emulators are then trained using Bayes Linear adjustment. On subsequent waves, the preliminary emulators should be passed to the function as the argument `previous_wave`.

The output consists of a list of four items: the preliminary emulators `base_emulators`, the trained emulators `emulators`, the next points to be put into the simulator `next_sample`, and the minimum enclosing hyperrectangle for the non-implausible region, given as ranges `new_ranges`.

Value

A list of base emulators, trained emulators for this wave, new sample points, and new ranges.

Examples

```
#ranges <- list(aSI = c(0.1, 0.8), aIR = c(0, 0.5), aSR = c(0, 0.05))
#outputs <- c('nS','nI','nR')
#targets <- list(
# list(val = 281, sigma = 10.43),
# list(val = 30, sigma = 11.16),
# list(val = 689, sigma = 14.32)
#)
#wave1 <- full_wave(GillespieSIR, GillespieValidation, ranges, outputs, targets,
# n_points = 30, deltas = rep(0.1, 3), quadratic = TRUE)
```

---

generate_new_runs *Generate Simulator Runs*

Description

A wrapper for a variety of sampling methods. Given a set of trained emulators, finds the next set of points that will be informative for the next wave of emulators.

Usage

```
generate_new_runs(
  emulators,
  ranges,
  n_points = 10 * length(ranges),
  z,
  method = "importance",
  include_line = TRUE,
  cutoff = 3,
  plausible_set,
  ...
)
```

Arguments

emulators	A list of Emulator objects, trained on the design points
ranges	The ranges of the input parameters
n_points	Optional. Specifies how many additional points are required. Default: 10*(number of emulators)
z	Checks implausibility of sample points to restrict to only non-implausible points.
method	Any of 'lhs', 'slice', 'optical'.
include_line	Should line sampling be applied after point generation? Default: TRUE.
cutoff	Optional. If z is given, this is the implausibility cutoff for the filtering. Default = 3
plausible_set	Optional - a set of non-implausible points from which to start.
...	Any parameters that need to be passed to a particular method (see below)

Details

If the method is 'lhs', this creates a new training set using LHS, and then finds the trace of the variance matrix of the emulators across these points (this is broadly equivalent to using V-optimality). We repeat this for n_runs, and select the configuration that minimises the mean of the variances across the emulators. If observations are given, then these are used to ensure that the new sample points are non-implausible according to the current emulators.

If the method is 'slice', then an initial point x must be provided. This is ideally an non-implausible point, or at least one close to the suspected non-implausible region. It then applies slice sampling, using implausibility as a measure of success.

If the method is 'optical', then the optical depth of the space in each parameter direction is calculated (using a known set of non-implausible points plausible_set), and used as a distribution for that parameter. Points are sampled from the collection of distributions and non-implausible points generated are filtered out. From the remaining points, a sample of the required size is generated using maximin criterion.

If the method is 'importance', importance sampling is used. Starting from a set of non-implausible (preferably space-filling) points, points are sampled from a distribution around the points, and included in the output based on a weighted measure gained from the mixture distribution of the initial points. The set plausible_set must be specified.

Note that the plausible_set parameter size differs between the two methods that use it (the 'optical' and 'importance') methods. The optical set should be as large as possible in order to accurately represent the optical depth in each parameter direction; the set for importance sampling should be smaller (and probably smaller than the desired number of output points) in order to expedite the initial set-up of the sampling strategy.

For any sampling strategy, the parameters emulators, ranges and z must be specified. If the method is 'slice', then the parameter x is necessary. All other parameters are optional.

After the first round of sampling, if line_sample is enabled, an exploration of the boundary of the non-implausible region is performed as follows. Two points are chosen at random, and a number of points are sampled uniformly along the line connecting these points. The sampled points are tested for implausibility, and (provided more than 50 of the points are non-implausible) the most separated of the points replace the two initial points.

Value

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

Examples

```
ranges <- list(aSI = c(0.1, 0.8), aIR = c(0, 0.5), aSR = c(0, 0.05))
ems <- emulator_from_data(GillespieSIR, output_names = c('nS', 'nI', 'nR'),
 ranges = ranges, quadratic = TRUE)
trained_ems <- purrr::map(seq_along(ems),
 ~ems[[.x]]$adjust(GillespieSIR, c('nS', 'nI', 'nR')[[.x]]))
targets <- list(
 list(val = 281, sigma = 10.43),
 list(val = 30, sigma = 11.16),
 list(val = 689, sigma = 14.32)
)
non_imp_points <- GillespieImplausibility[GillespieImplausibility$I <= 4, names(ranges)]
example_point <- unlist(non_imp_points[sample(1:length(non_imp_points[,1]), 1),],
 use.names = FALSE)
pts_lhs <- generate_new_runs(trained_ems, ranges, 10, targets, cutoff = 3)
#pts_slice <- generate_new_runs(trained_ems, ranges, 10, targets,
 #method = 'slice', cutoff = 4, x = example_point)
pts_optical <- generate_new_runs(trained_ems, ranges, 10, targets,
 method = 'optical', cutoff = 4, plausible_set = non_imp_points)
non_imp_sample <- non_imp_points[sample(seq_along(non_imp_points[,1]), 20),]
pts_importance <- generate_new_runs(trained_ems, ranges, 10, targets,
 method = 'importance', cutoff = 4, plausible_set = non_imp_sample, include_line = FALSE)
```

---

get_linear_model *Linear Model Generation*

Description

Creates a linear model from data and a list of inputs. There are two ways to generate the model; either start with all possible linear terms, and then stepwise remove them (using `step`), or start with an intercept add stepwise add linear terms, only retaining them if the AIC is improved. Which is chosen is dependent on the value of `add`; in the event where `add = FALSE` and there are not enough degrees of freedom, a warning will be given.

Usage

```
get_linear_model(data, ranges, output_name, add = FALSE)
```

Arguments

`data`	A `data.frame` containing the input and output values
`ranges`	A named `list` consisting of the ranges of the input parameters
`output_name`	A string corresponding to the output to be modelled
`add`	Should we perform stepwise add or stepwise delete? Default: `FALSE`

Value

The fitted model

get_quadratic_model *Quadratic Model Generation*

Description

Creates a quadratic model from data and a list of inputs. There are two ways to generate the model; either start with all possible linear and quadratic terms, and then stepwise remove them (using `step`), or start with a linear model (maybe from `get_linear_model`) and add quadratic terms one by one, only retaining them if the AIC is improved. Which is chosen is dependent on the value of `add`; in the event where `add = FALSE` and there are not enough degrees of freedom, a warning will be given.

Usage

```
get_quadratic_model(
  data,
  ranges,
  output_name,
  add = FALSE,
  linear_model = NULL
)
```

Arguments

`data`	A `data.frame` containing the input and output values
`ranges`	A named `list` consisting of the ranges of the input parameters
`output_name`	A string corresponding to the output to be modelled
`add`	Should we perform stepwise add or stepwise delete? Default: `FALSE`
`linear_model`	Optional. A linear model to augment if add = TRUE

Value

The fitted model

GillespieImplausibility *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 `GillespieSIR` dataset) for each of the outputs nS, nI, nR, and the maximum implausibility is included. The target values used in calculating implausibility were: nS: 281 (sigma 10.43); nI: 30 (sigma 11.16); nR: 689 (sigma 14.32)

Usage

```
GillespieImplausibility
```

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

GillespieMultiWaveData

Sample Multi-wave Results

Description

An rda object containing four data.frames: an initial set of points (equivalent to `rbind(GillespieSIR,GillespieValida`
and the 90 points generated at each of three subsequent waves. The trained emulators are provided
in `GillespieMultiWaveEmulators`.

Usage

```
GillespieMultiWaveData
```

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

GillespieMultiWaveEmulators

Sample Multi-wave Emulators

Description

An rda object containing three waves of emulation on the Gillespie SIR model.

Usage

```
GillespieMultiWaveEmulators
```

Format

A list containing `Emulator` objects:

Wave 1 Emulators trained on GillespieSIR to generate wave 2 points

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

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

`GillespieSIR` *Sample SIR data*

Description

A small dataset containing points generated using the Gillespie algorithm. The SIR 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=20.

Usage

```
GillespieSIR
```

Format

A data frame with 30 rows and 6 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 Final number of S

nI Final number of I

nR Final number of R

`GillespieValidation` *Sample SIR validation data*

Description

A small dataset containing points generated using the Gillespie algorithm. Very similar to `GillespieSIR`, slightly larger in size.

Usage

```
GillespieValidation
```

Format

A data frame with 60 rows and 6 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 Final number of S

nI Final number of I

nR Final number of R

---

min_implausibility *Minimum Implausibilities on a Lattice*

Description

Calculates minimum implausibility of a set of emulators, given target values. Implausibility is calculated for each (univariate) emulator and the corresponding target value, and the n-th implausibility is computed. This occurs in a lattice hypercube of size points_per_dim^n, and all points are put into bins in the two target directions. The minimum is the smallest such value that occurs in each bin.

Usage

```
min_implausibility(
  targets,
  ranges,
  points_per_dim = 50,
  plot_vars = names(ranges)[1:2],
  emulators = NULL,
  imps = NULL,
  ...
)
```

Arguments

`targets`	The target values of the outputs, in the normal form
`ranges`	The ranges of the input variables, as a named list
`points_per_dim`	The number of lattice points in each input direction
`plot_vars`	A list of two names, or a single name, corresponding to the plotting inputs
`emulators`	A set of `Emulator` objects
`imps`	If implausibility has already been calculated across a grid, the `data.frame` goes here
`...`	Any options to be passed to the `nth_implausible` function.

Details

One or two target directions can be specified.

Value

A `data.frame` corresponding to the bins and proportions.

Examples

```
ranges <- list(aSI = c(0.1, 0.8), aIR = c(0, 0.5), aSR = c(0, 0.05))
out_vars <- c('nS', 'nI', 'nR')
targets <- list(
 list(val = 281, sigma = 37.26),
 list(val = 30, sigma = 11.16),
 list(val = 689, sigma = 31.72))
first_ems <- emulator_from_data(GillespieSIR, output_names = out_vars,
 ranges = ranges, quadratic = TRUE)
trained_ems <- purrr::map(seq_along(first_ems),
 ~first_ems[[.x]]$adjust(GillespieSIR, out_vars[[.x]]))
min_implausibility(targets, ranges, 5, emulators = trained_ems)
min_implausibility(targets, ranges, 5, plot_vars = c('aIR', 'aSR'),
emulators = trained_ems, n = 2, max_imp = 30)
```

---

nth_implausible *n-th Maximum Implausibility*

Description

For a collection of emulators, it can be helpful to combine the implausibility measures for a given observation. The maximum implausibility is, simply, the largest implausibility value given by the emulators for each output; the 2nd maximum is the maximum of the set without the maximum, and so on.

Usage

```
nth_implausible(emulators, x, z, n = 1, max_imp = 20)
```

Arguments

`emulators`	A set of `Emulator` objects.
`x`	An input point
`z`	The observed outputs, either as a numeric vector or as a collection of val, sigma pairs (see examples)
`n`	The implausibility level to return. By default, the median implausibility is chosen
`max_imp`	A maximum implausibility to consider: in most cases, it is useful to truncate the size of the I(x). Default: 20.

Value

The n-th maximum implausibility value.

Examples

```
ems <- emulator_from_data(GillespieSIR, output_names = c('nS', 'nI', 'nR'),
 ranges = list(aSI = c(0.1, 0.8), aIR = c(0, 0.5), aSR = c(0, 0.05)),
 quadratic = TRUE)
targets <- list(
 list(val = 281, sigma = 10.43),
 list(val = 30, sigma = 11.16),
 list(val = 689, sigma = 14.32)
)
nth_implausible(ems, c(0.4, 0.25, 0.025), targets)
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)
)
nth_implausible(ems, grid, targets, n = 2)
```

---

optical_depth *Optical Depth Of Emulator Implausibility*

Description

Calculates optical depth of the implausibility of a set of emulators, given target values. Implausibility is calculated for each (univariate) emulator and the corresponding target value, and the n-th implausibility is computed. This occurs in a lattice hypercube of size points_per_dim^n, and all points are put into bins in the two target directions. The optical depth is the proportion of points in a given bin that are non-implausible.

Usage

```
optical_depth(
  targets,
  ranges,
  points_per_dim,
  plot_vars = names(ranges)[1:2],
  emulators = NULL,
  imps = NULL,
  cutoff = 3,
  ...
)
```

Arguments

targets	The target values of the outputs, in the normal form
ranges	The ranges of the input variables, as a named list
points_per_dim	The number of lattice points in each input direction
plot_vars	A list of two names, or a single name, corresponding to the plotting inputs
emulators	A set of Emulator objects
imps	If implausibility has already been calculated across a grid, the data.frame goes here
cutoff	The implausibility cutoff. Default: 3
...	Any options to be passed to the nth_implausible function.

Details

One or two target directions can be specified.

Value

A `data.frame` corresponding to the bins and proportions.

Examples

```
ranges <- list(aSI = c(0.1, 0.8), aIR = c(0, 0.5), aSR = c(0, 0.05))
out_vars <- c('nS', 'nI', 'nR')
targets <- list(
 list(val = 281, sigma = 37.26),
 list(val = 30, sigma = 11.16),
 list(val = 689, sigma = 31.72))
first_ems <- emulator_from_data(GillespieSIR, output_names = out_vars,
 ranges = ranges, quadratic = TRUE)
trained_ems <- purrr::map(seq_along(first_ems),
 ~first_ems[[.x]]$adjust(GillespieSIR, out_vars[[.x]]))
optical_depth(targets, ranges, 5, emulators = trained_ems)
optical_depth(targets, ranges, 5, plot_vars = c('aIR', 'aSR'),
emulators = trained_ems, cutoff = 4, n = 2, max_imp = 30)
optical_depth(targets, ranges, 5, plot_vars = 'aSR',
emulators = trained_ems, cutoff = 2, n = 2)
```

`output_plot` *Emulator Plots with Outputs*

Description

Plots emulator outputs across a set of points, with the corresponding observations overlaid (with the appropriate uncertainty). If no points are provided, then `npoints` points are uniformly sampled from the input region. Else the provided points are used - for example, if a non-implausible space is known.

Usage

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

Arguments

`emulators`	A list of `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. Default: `NULL`
`npoints`	If no points provided, how many input points to evaluate? Default: 1000

Value

A ggplot object.

Examples

```
ranges <- list(aSI = c(0.1, 0.8), aIR = c(0, 0.5), aSR = c(0, 0.05))
outputs <- c('nS','nI','nR')
targets <- list(
 nS = list(val = 281, sigma = 10.43),
 nI = list(val = 30, sigma = 11.16),
 nR = list(val = 689, sigma = 14.32)
)
ems <- emulator_from_data(GillespieSIR, outputs, ranges,
 deltas = rep(0.1, 3), quadratic = TRUE)
t_ems <- purrr::map(seq_along(ems), ~ems[[.]]$adjust(GillespieSIR, outputs[.]))
output_plot(t_ems, targets)
```

---

plot_implausible *Implausibility Plots*

Description

Generates implausibility plots: either optical depth, or minimum implausibility. Simply a plot wrapper to make later things easier for lattice plots, etc. However, it does have value in and of itself.

Usage

```
plot_implausible(df, names, nvars, min = FALSE, ticks = FALSE)
```

Arguments

df	A data.frame containing the inputs and implausibilities
names	A list of names for the inputs.
nvars	The number of variables in the plotting region: either 1 or 2.
min	Should minimum implausibility be plotted? Default: FALSE
ticks	Are axis ticks required? Default: FALSE

Value

A ggplot object

Examples

```
ranges <- list(aSI = c(0.1, 0.8), aIR = c(0, 0.5), aSR = c(0, 0.05))
implaus <- GillespieImplausibility[,c(names(ranges), "I")]
targets <- list(
 list(val = 281, sigma = 10.43),
 list(val = 30, sigma = 11.16),
 list(val = 689, sigma = 14.32)
)
op_depth <- optical_depth(targets, ranges, 20, plot_vars = c('aSI','aIR'), imps = implaus)
plot_implausible(op_depth, c('aSI','aIR'), 2, ticks = TRUE)
min_imp <- min_implausibility(targets, ranges, points_per_dim = 20, imps = implaus)
plot_implausible(min_imp, c('aSI','aIR'), 2, min = TRUE, ticks = TRUE)
```

plot_lattice *Implausibility Plot Lattice*

Description

Creates a lattice of plots, consisting of optical depths (both univariate and two-variable projections), and minimum implausibility. It is advisable to pass no more than 6 input parameter names to this function, as the graphs will be too small for meaningful analysis if more are provided.

Usage

```
plot_lattice(implausibilities, targets, ranges, np = 50, ...)
```

Arguments

implausibilities	A list of implausibilities for input points
targets	A list of observations, given in the usual form
ranges	The ranges of each of the input parameters.
np	The number of bins in each part of the plot grids
...	Any additional inputs (to be passed to optical_depth)

Value

A ggplot object

Examples

```
imp_data <- GillespieImplausibility
ranges <- list(aSI = c(0.1, 0.8), aIR = c(0, 0.5), aSR = c(0, 0.05))
targets <- list(
 list(val = 281, sigma = 10.43),
 list(val = 30, sigma = 11.16),
 list(val = 689, sigma = 14.32)
)
plot_lattice(imp_data[,c(names(ranges), "I")], targets, ranges, np = 20)
```

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))
)
```

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.

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.

Examples

```
targets <- list(
 nS = list(val = 281, sigma = 10.43),
 nI = list(val = 30, sigma = 11.16),
 nR = list(val = 689, sigma = 14.32)
)
simulator_plot(GillespieMultiWaveData, targets)
simulator_plot(GillespieMultiWaveData[2:4], targets,
 zero_in = FALSE, wave_numbers = c(1,3))
```

---

space_removed *Space Removal*

Description

Finds the proportion of space removed as a function of implausibility cut-off, and of structural discrepancy, or changed variance.

Usage

```
space_removed(
  emulators,
  validation_points,
  z,
  n_points = 10,
  u_mod = seq(0.8, 1.2, by = 0.1),
  intervals = seq(0, 10, length.out = 200),
  modified = "disc"
)
```

Arguments

emulators	A set of Emulator objects.
validation_points	The validation set used in this wave.
z	The observations with which to match, as list(val,sigma) pairs.
n_points	The number of points in each dimension of the grid.
u_mod	The percentage differences in structural discrepancy to examine.
intervals	The set of implausibility cut-offs to consider.
modified	What should be changed: model discrepancy (disc) or emulator variance (var)?

Details

The reduction in space is found by evaluating over a p^d regular grid, where p is chosen by n_points and d is the dimension of the input space. Larger values of n_points will give a more accurate reflection of removed space, at high computational cost. For the purpose of quick diagnostics, n_points = 5 is acceptable.

Value

A list of two data.frames, one for removed space and one for misclassifications.

Examples

```
ranges <- list(aSI = c(0.1, 0.8), aIR = c(0, 0.5), aSR = c(0, 0.05))
targets <- list(
 list(val = 281, sigma = 10.43),
 list(val = 30, sigma = 11.16),
 list(val = 689, sigma = 14.32)
)
outputs <- c('nS','nI','nR')
ems <- emulator_from_data(GillespieSIR, outputs, ranges, deltas = rep(0.1, 3), quadratic = TRUE)
t_ems <- purrr::map(seq_along(ems), ~ems[[.]]$adjust(GillespieSIR, outputs[[.]]))
names(t_ems) <- outputs
removal <- space_removed(ems, GillespieValidation, targets,
 n_points = 5, u_mod = seq(0.75, 1.25, by = 0.25), intervals = seq(2, 6, by = 0.1))
```

`standard_errors` *Emulator Standard Errors*

Description

Finds and plots emulator standard errors. For an emulator of a simulator function `f(x)`, and a validation data set `X`, finds the standard errors in the form `(f(x)-E[f(x)])/sqrt(Var[f(x)])`, where `E[f(x)]` is the emulator expectation, and `Var[f(x)]` is the emulator variance, at each point `x` in `X`.

Usage

```
standard_errors(
  emulator,
  input_points,
  output_points,
  output_name,
  plt = T,
  ...
)
```

Arguments

`emulator`	An `Emulator` object.
`input_points`	A set of validation points.
`output_points`	The outputs, `f(x)`, from the simulator.
`output_name`	Optional. A name for the output.
`plt`	Should a plot be shown (default: T).
`...`	Dummy parameters (for compatibility with diagnostic wrapper)

Value

A list of standard errors.

Examples

```
em <- emulator_from_data(GillespieSIR, output_names = c('nS', 'nI', 'nR'),
 ranges = list(aSI = c(0.1, 0.8), aIR = c(0, 0.5), aSR = c(0, 0.05)),
 quadratic = TRUE)[[1]]
standard_errors(em, GillespieValidation[,1:3], GillespieValidation[,'nS'], 'nS')
#> (0.7864384, 0.01426296, 0.001072935)
```

validation_diagnostics *Emulator Diagnostics*

Description

Plots standard diagnostics for emulators.

Usage

```
validation_diagnostics(
  emulators,
  validation_points,
  output_names,
  which_diag = "all",
  targets = NULL,
  ...
)
```

Arguments

emulators	A list of emulators on which to perform diagnostics
validation_points	The validation set
output_names	The list of outputs to perform diagnostics on
which_diag	Which diagnostics should be performed?
targets	If required, the list of observations for the outputs
...	Any additional parameters to pass to the diagnostic tests.

Details

These diagnostics are based on having two datasets: a training set and a validation set. The emulators will have been trained on the training set, and the validation set is passed to the functions in this wrapper.

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

Standard Errors (se)

Comparison Diagnostics (cd)

Classification Error (ce)

All of the above (all)

For details on each of these, see the help files for standard_errors, comparison_diagnostics and classification_error respectively.

Value

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

Examples

```
output_names <- c('nS','nI','nR')
ems <- emulator_from_data(GillespieSIR, output_names,
 ranges = list(aSI = c(0.1, 0.8), aIR = c(0, 0.5), aSR = c(0, 0.05)),
 quadratic = TRUE)
targets <- list(
 list(val = 281, sigma = 10.43),
 list(val = 30, sigma = 11.16),
 list(val = 689, sigma = 14.32)
)
validation_diagnostics(ems, GillespieValidation, output_names, targets = targets)
validation_diagnostics(ems, GillespieValidation, output_names, c('se','cd'))
validation_diagnostics(ems[1:2], GillespieValidation, output_names[1:2], 'ce', targets[1:2])
validation_diagnostics(ems, GillespieValidation, output_names,
 targets = targets, sd = 1, cutoff = 4)
```

---

validation_pairs *Validation Set Comparisons and Implausibility*

Description

Creates pairs plots on the set of validation points.

Usage

```
validation_pairs(ems, validation_points, z, orig_ranges, ...)
```

Arguments

ems	The list of trained emulators
validation_points	The validation set to be plotted
z	The target values for each emulated output
orig_ranges	The original ranges for the input parameters (if desired)
...	Any additional parameters to be passed to internal functions.

Details

Plots are organised as:

a) Emulated vs Simulator Output (lower diagonal). The emulator outputs are compared against the simulator outputs. Points whose emulated output lies outside the 3-sigma region of the simulated output are coloured red; those inside are coloured green; a gradient between the two extremes indicates goodness-of-fit;

b) Implausibility (upper diagonal). The implausibility for each point is calculated, using the same colour scaling as the lower diagonal.

Value

A data.frame containing the validation points, with goodness-of-fit and implausibility.

Examples

```
ems <- emulator_from_data(GillespieSIR, c('nS','nI','nR'),
 ranges = list(aSI = c(0.1, 0.8), aIR = c(0, 0.5), aSR = c(0, 0.05)),
 quadratic = TRUE)
targets <- list(
 list(val = 281, sigma = 10.43),
 list(val = 30, sigma = 11.16),
 list(val = 689, sigma = 14.32)
)
validation_pairs(ems, GillespieValidation, targets)
```

---

validation_plots *Validation Set Plotting*

Description

Plots each of the emulator outputs against each input. For each input parameter, the emulator expectation is plotted for each output. These plots are presented as a set, to better identify trends and dependencies in the emulators outputs.

Usage

```
validation_plots(emulators, input_points, output_names)
```

Arguments

emulators	A set of Emulator objects.
input_points	The validation input points on which to evaluate the emulators
output_names	The names of the output parameters.

Examples

```
ems <- emulator_from_data(GillespieSIR, output_names = c('nS', 'nI', 'nR'),
 ranges = list(aSI = c(0.1, 0.8), aIR = c(0, 0.5), aSR = c(0, 0.05)),
 quadratic = TRUE)
validation_plots(ems, GillespieValidation[,1:3], c('nS', 'nI', 'nR'))
```

---

wave_points *Plot Points of Waves*

Description

Creates a set of pairs plots of the input points for multiple waves.

Usage

```
wave_points(pts_list, in_names)
```

Arguments

pts_list	A list object, whose elements are data.frames of points
in_names	The input dimension names.

Details

For each pair of inputs, the points from a succession of waves are plotted, coloured according to the wave number. Histograms of the points for each input (broadly interpreted as marginal distributions for the inputs) are provided on the main diagonal.

Value

A ggplot object.

---

wave_variance *Emulator Variance across waves*

Description

Plots the emulator variance for each output across emulator waves.

Usage

```
wave_variance(
  waves,
  output_names,
  plot_dirs = names(waves[[1]][[1]]$ranges)[1:2],
  wave_numbers = 1:length(waves),
  n_points = 40,
  sd = FALSE
)
```

Arguments

waves	A list of lists of Emulator objects, corresponding to the waves
output_names	The list of desired outputs to be plotted
plot_dirs	The (two) input parameters to be plotted.
wave_numbers	A numeric vector of which waves to plot.
n_points	The number of grid points per plotting dimension. Default: 20
sd	Should the standard deviation be plotted instead of the variance? Default: FALSE

Details

It is instructive to look at the change in emulator variance over successive waves, rather than across successive outputs. This function provides a means of doing so quickly for each emulator output.

A 2d slice is taken across the input space, where mid-range values of any non-plotted parameters are fixed. The emulator variance (or standard deviation) is then calculated across the two-dimensional subspace for each wave, and for each output.

Value

A list of data.frames, each corresponding to a given output over waves.

Examples

```
outputs <- c('nS', 'nI', 'nR')
em_var <- wave_variance(GillespieMultiWaveEmulators, outputs, n_points = 5)
em_sd <- wave_variance(GillespieMultiWaveEmulators, c('nI', 'nR'),
 plot_dirs = c('aIR', 'aSR'), n_points = 5, sd = TRUE)
```

Index