Package: dgpsi (via r-universe)

September 2, 2024

Type Package

Title Interface to 'dgpsi' for Deep and Linked Gaussian Process Emulations

Version 2.4.0-9000

Maintainer Deyu Ming <deyu.ming.16@ucl.ac.uk>

Description Interface to the 'python' package 'dgpsi' for Gaussian process, deep Gaussian process, and linked deep Gaussian process emulations of computer models and networks using stochastic imputation (SI). The implementations follow Ming & Guillas (2021) <doi:10.1137/20M1323771> and Ming, Williamson, & Guillas (2023) <doi:10.1080/00401706.2022.2124311> and Ming & Williamson (2023) <arXiv:2306.01212>. To get started with the package, see <https://mingdeyu.github.io/dgpsi-R/>.

License MIT + file LICENSE

URL https://github.com/mingdeyu/dgpsi-R, https://mingdeyu.github.io/dgpsi-R/

BugReports https://github.com/mingdeyu/dgpsi-R/issues

Encoding UTF-8

Depends R (>= 4.0)

Imports reticulate (>= 1.26), benchmarkme (>= 1.0.8), utils, ggplot2, ggforce, reshape2, patchwork, lhs, methods, stats, bitops, clhs, dplyr, uuid

Suggests knitr, rmarkdown, MASS, R.utils, spelling

LazyData false

VignetteBuilder knitr

RoxygenNote 7.2.1

Language en-US

Repository https://mingdeyu.r-universe.dev

RemoteUrl https://github.com/mingdeyu/dgpsi-r

RemoteRef HEAD

RemoteSha 1988310d7c17489a91294d5465ce201b572f8de2

Contents

`alm` *Locate the next design point for a (D)GP emulator or a bundle of (D)GP emulators using ALM*

Description

This function searches from a candidate set to locate the next design point(s) to be added to a (D)GP emulator or a bundle of (D)GP emulators using the Active Learning MacKay (ALM), see the reference below.

Usage

```
alm(object, x_cand, ...)

## S3 method for class 'gp'
alm(object, x_cand, batch_size = 1, M = 50, workers = 1, ...)

## S3 method for class 'dgp'
alm(object, x_cand, batch_size = 1, M = 50, workers = 1, aggregate = NULL, ...)

## S3 method for class 'bundle'
alm(object, x_cand, batch_size = 1, M = 50, workers = 1, aggregate = NULL, ...)
```

Arguments

`object`	can be one of the following: • the S3 class `gp`. • the S3 class `dgp`. • the S3 class `bundle`.
`x_cand`	a matrix (with each row being a design point and column being an input dimension) that gives a candidate set from which the next design point(s) are determined. If `object` is an instance of the `bundle` class, `x_cand` could also be a list with the length equal to the number of emulators contained in the `object`. Each slot in `x_cand` is a matrix that gives a candidate set for each emulator included in the bundle. See *Note* section below for further information.
`...`	any arguments (with names different from those of arguments used in `alm()`) that are used by `aggregate` can be passed here.
`batch_size`	an integer that gives the number of design points to be chosen. Defaults to `1`.
`M`	the size of the conditioning set for the Vecchia approximation in the criterion calculation. This argument is only used if the emulator `object` was constructed under the Vecchia approximation. Defaults to `50`.
`workers`	the number of processes to be used for the criterion calculation. If set to `NULL`, the number of processes is set to `max physical cores available %/% 2`. Defaults to `1`.
`aggregate`	an R function that aggregates scores of the ALM across different output dimensions (if `object` is an instance of the `dgp` class) or across different emulators (if `object` is an instance of the `bundle` class). The function should be specified in the following basic form: • the first argument is a matrix representing scores. The rows of the matrix correspond to different design points. The number of columns of the matrix equals to:

  - the emulator output dimension if `object` is an instance of the `dgp` class; or
  - the number of emulators contained in `object` if `object` is an instance of the `bundle` class.
- the output should be a vector that gives aggregations of scores at different design points.

Set to `NULL` to disable the aggregation. Defaults to `NULL`.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

- If `object` is an instance of the `gp` class, a vector is returned with the length equal to `batch_size`, giving the positions (i.e., row numbers) of next design points from `x_cand`.
- If `object` is an instance of the `dgp` class, a matrix is returned with row number equal to `batch_size` and column number equal to one (if `aggregate` is not `NULL`) or the output dimension (if `aggregate` is `NULL`), giving positions (i.e., row numbers) of next design points from `x_cand` to be added to the DGP emulator across different outputs. If `object` is a DGP emulator with either `Hetero` or `NegBin` likelihood layer, the returned matrix has two columns with the first column giving positions of next design points from `x_cand` that correspond to the mean parameter of the normal or negative Binomial distribution, and the second column giving positions of next design points from `x_cand` that correspond to the variance parameter of the normal distribution or the dispersion parameter of the negative Binomial distribution.
- If `object` is an instance of the `bundle` class, a matrix is returned with row number equal to `batch_size` and column number equal to the number of emulators in the bundle, giving positions (i.e., row numbers) of next design points from `x_cand` to be added to individual emulators.

Note

- The column order of the first argument of `aggregate` must be consistent with the order of emulator output dimensions (if `object` is an instance of the `dgp` class), or the order of emulators placed in `object` if `object` is an instance of the `bundle` class;
- If `x_cand` is supplied as a list when `object` is an instance of `bundle` class and a `aggregate` function is provided, the matrices in `x_cand` must have common rows (i.e., the candidate sets of emulators in the bundle have common input locations) so the `aggregate` function can be applied.
- Any R vector detected in `x_cand` will be treated as a column vector and automatically converted into a single-column R matrix.

References

MacKay, D. J. (1992). Information-based objective functions for active data selection. *Neural Computation*, **4(4)**, 590-604.

Examples

```
## Not run:

# load packages and the Python env
library(lhs)
library(dgpsi)

# construct a 1D non-stationary function
f <- function(x) {
 sin(30*((2*x-1)/2-0.4)^5)*cos(20*((2*x-1)/2-0.4))
}

# generate the initial design
X <- maximinLHS(10,1)
Y <- f(X)

# training a 2-layered DGP emulator with the global connection off
m <- dgp(X, Y, connect = F)

# generate a candidate set
x_cand <- maximinLHS(200,1)

# locate the next design point using ALM
next_point <- alm(m, x_cand = x_cand)
X_new <- x_cand[next_point,,drop = F]

# obtain the corresponding output at the located design point
Y_new <- f(X_new)

# combine the new input-output pair to the existing data
X <- rbind(X, X_new)
Y <- rbind(Y, Y_new)

# update the DGP emulator with the new input and output data and refit
m <- update(m, X, Y, refit = TRUE)

# plot the LOO validation
plot(m)

## End(Not run)
```

`combine` *Combine layers*

Description

This function combines customized layers into a DGP or linked (D)GP structure.

Usage

```
combine(...)
```

Arguments

`...` a sequence of lists:

1. For DGP emulations, each list represents a DGP layer and contains GP nodes (produced by `kernel()`), or likelihood nodes (produced by `Poisson()`, `Hetero()`, or `NegBin()`).
2. For linked (D)GP emulations, each list represents a system layer and contains emulators (produced by `gp()` or `dgp()`) in that layer.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

A list defining a DGP structure (for `struc` of `dgp()`) or a linked (D)GP structure (for `struc` for `lgp()`).

Examples

```
## Not run:

# See lgp() for an example.

## End(Not run)
```

`continue` *Continue the training of a DGP emulator*

Description

This function implements additional training iterations for a DGP emulator.

Usage

```
continue(
  object,
  N = NULL,
  cores = 1,
  ess_burn = 10,
  verb = TRUE,
  burnin = NULL,
  B = NULL
)
```

Arguments

object	an instance of the dgp class.
N	additional number of iterations for the DGP emulator training. If set to NULL, the number of iterations is set to 500 if the DGP emulator was constructed without the Vecchia approximation, and is set to 200 if Vecchia approximation was used. Defaults to NULL.
cores	the number of processes to be used to optimize GP components (in the same layer) at each M-step of the training. If set to NULL, the number of processes is set to (max physical cores available - 1) if the DGP emulator was constructed without the Vecchia approximation. Otherwise, the number of processes is set to max physical cores available %/% 2. Only use multiple processes when there is a large number of GP components in different layers and optimization of GP components is computationally expensive. Defaults to 1.
ess_burn	number of burnin steps for the ESS-within-Gibbs at each I-step of the training. Defaults to 10.
verb	a bool indicating if the progress bar will be printed during the training: 1. FALSE: the training progress bar will not be displayed. 2. TRUE: the training progress bar will be displayed. Defaults to TRUE.
burnin	the number of training iterations to be discarded for point estimates calculation. Must be smaller than the overall training iterations so-far implemented. If this is not specified, only the last 25% of iterations are used. This overrides the value of burnin set in dgp(). Defaults to NULL.
B	the number of imputations to produce the predictions. Increase the value to account for more imputation uncertainties. This overrides the value of B set in dgp() if B is not NULL. Defaults to NULL.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

An updated object.

Note

- One can also use this function to fit an untrained DGP emulator constructed by dgp() with training = FALSE.
- The following slots:
  - loo and oos created by validate(); and
  - results created by predict() in object will be removed and not contained in the returned object.

Examples

```
## Not run:

# See dgp() for an example.

## End(Not run)
```

`design` *Sequential design of a (D)GP emulator or a bundle of (D)GP emulators*

Description

This function implements the sequential design of a (D)GP emulator or a bundle of (D)GP emulators.

Usage

```
design(
  object,
  N,
  x_cand,
  y_cand,
  n_cand,
  limits,
  int,
  f,
  reps,
  freq,
  x_test,
  y_test,
  reset,
  target,
  method,
  eval,
  verb,
  autosave,
  new_wave,
  M_val,
  cores,
  ...
)

## S3 method for class 'gp'
design(
  object,
  N,
```

```
  x_cand = NULL,
  y_cand = NULL,
  n_cand = 200,
  limits = NULL,
  int = FALSE,
  f = NULL,
  reps = 1,
  freq = c(1, 1),
  x_test = NULL,
  y_test = NULL,
  reset = FALSE,
  target = NULL,
  method = vigf,
  eval = NULL,
  verb = TRUE,
  autosave = list(),
  new_wave = TRUE,
  M_val = 50,
  cores = 1,
  ...
)

## S3 method for class 'dgp'
design(
  object,
  N,
  x_cand = NULL,
  y_cand = NULL,
  n_cand = 200,
  limits = NULL,
  int = FALSE,
  f = NULL,
  reps = 1,
  freq = c(1, 1),
  x_test = NULL,
  y_test = NULL,
  reset = FALSE,
  target = NULL,
  method = vigf,
  eval = NULL,
  verb = TRUE,
  autosave = list(),
  new_wave = TRUE,
  M_val = 50,
  cores = 1,
  train_N = NULL,
  refit_cores = 1,
  pruning = TRUE,
```

```
  control = list(),
  ...
)

## S3 method for class 'bundle'
design(
  object,
  N,
  x_cand = NULL,
  y_cand = NULL,
  n_cand = 200,
  limits = NULL,
  int = FALSE,
  f = NULL,
  reps = 1,
  freq = c(1, 1),
  x_test = NULL,
  y_test = NULL,
  reset = FALSE,
  target = NULL,
  method = vigf,
  eval = NULL,
  verb = TRUE,
  autosave = list(),
  new_wave = TRUE,
  M_val = 50,
  cores = 1,
  train_N = NULL,
  refit_cores = 1,
  ...
)
```

Arguments

object	can be one of the following: • the S3 class gp. • the S3 class dgp. • the S3 class bundle.
N	the number of steps for the sequential design.
x_cand	a matrix (with each row being a design point and column being an input dimension) that gives a candidate set in which the next design point is determined. If x_cand = NULL, the candidate set will be generated using n_cand, limits, and int. Defaults to NULL.
y_cand	a matrix (with each row being a simulator evaluation and column being an output dimension) that gives the realizations from the simulator at input positions in x_cand. Defaults to NULL.
n_cand	an integer that gives

- the size of the candidate set in which the next design point is determined, if `x_cand = NULL`;
- the size of a sub-set to be sampled from the candidate set `x_cand` at each step of the sequential design to determine the next design point, if `x_cand` is not `NULL`.

Defaults to `200`.

`limits` a two-column matrix that gives the ranges of each input dimension, or a vector of length two if there is only one input dimension. If a vector is provided, it will be converted to a two-column row matrix. The rows of the matrix correspond to input dimensions, and its first and second columns correspond to the minimum and maximum values of the input dimensions. Set `limits = NULL` if `x_cand` is supplied. This argument is only used when `x_cand` is not supplied, i.e., `x_cand = NULL`. Defaults to `NULL`.

`int` a bool or a vector of bools that indicates if an input dimension is an integer type. If a bool is given, it will be applied to all input dimensions. If a vector is provided, it should have a length equal to the input dimensions and will be applied to individual input dimensions. Defaults to `FALSE`.

`f` an R function that represents the simulator. `f` needs to be specified with the following basic rules:

- the first argument of the function should be a matrix with rows being different design points and columns being input dimensions.
- the output of the function can either
  - a matrix with rows being different outputs (corresponding to the input design points) and columns being output dimensions. If there is only one output dimension, the matrix still needs to be returned with a single column.
  - a list with the first element being the output matrix described above and, optionally, additional named elements which will update values of any arguments with the same names passed via `...`. The list output can be useful if some additional arguments of `f` and `aggregate` need to be updated after each step of the sequential design.

See *Note* section below for further information. This argument is used when `y_cand = NULL`. Defaults to `NULL`.

`reps` an integer that gives the number of repetitions of the located design points to be created and used for evaluations of `f`. Set the argument to an integer greater than `1` if `f` is a stochastic function that can generate different responses given a same input and the supplied emulator `object` can deal with stochastic responses, e.g., a (D)GP emulator with `nugget_est = TRUE` or a DGP emulator with a likelihood layer. The argument is only used when `f` is supplied. Defaults to `1`.

`freq` a vector of two integers with the first element giving the frequency (in number of steps) to re-fit the emulator, and the second element giving the frequency to implement the emulator validation (for RMSE). Defaults to `c(1, 1)`.

`x_test` a matrix (with each row being an input testing data point and each column being an input dimension) that gives the testing input data to evaluate the emulator after each step of the sequential design. Set to `NULL` for the LOO-based emulator validation. Defaults to `NULL`. This argument is only used if `eval = NULL`.

`y_test` the testing output data that correspond to `x_test` for the emulator validation after each step of the sequential design:

- if `object` is an instance of the gp class, `y_test` is a matrix with only one column and each row being an testing output data point.
- if `object` is an instance of the dgp class, `y_test` is a matrix with its rows being testing output data points and columns being output dimensions.

Set to `NULL` for the LOO-based emulator validation. Defaults to `NULL`. This argument is only used if `eval = NULL`.

`reset` a bool or a vector of bools indicating whether to reset hyperparameters of the emulator to their initial values when it was initially constructed after the input-output update and before the re-fit. If a bool is given, it will be applied to every step of the sequential design. If a vector is provided, its length should be equal to `N` and will be applied to individual steps of the sequential design. Defaults to `FALSE`.

`target` a numeric or a vector that gives the target RMSEs at which the sequential design is terminated. Defaults to `NULL`, in which case the sequential design stops after `N` steps. See *Note* section below for further information about `target`.

`method` an R function that give indices of designs points in a candidate set. The function must satisfy the following basic rules:

- the first argument is an emulator object that can be either an instance of
  - the gp class (produced by `gp()`);
  - the dgp class (produced by `dgp()`);
  - the `bundle` class (produced by `pack()`).
- the second argument is a matrix with rows representing a set of different design points.
- the output of the function
  - is a vector of indices if the first argument is an instance of the gp class;
  - is a matrix of indices if the first argument is an instance of the dgp class. If there are different design points to be added with respect to different outputs of the DGP emulator, the column number of the matrix should equal to the number of the outputs. If design points are common to all outputs of the DGP emulator, the matrix should be single-columned. If more than one design points are determined for a given output or for all outputs, the indices of these design points are placed in the matrix with extra rows.
  - is a matrix of indices if the first argument is an instance of the `bundle` class. Each row of the matrix gives the indices of the design points to be added to individual emulators in the bundle.

See `alm()`, `mice()`, `pei()`, and `vigf()` for examples on customizing `method`. Defaults to `vigf()`.

`eval` an R function that calculates the customized evaluating metric of the emulator. The function must satisfy the following basic rules:

- the first argument is an emulator object that can be either an instance of
  - the gp class (produced by `gp()`);

  - the dgp class (produced by dgp());
  - the bundle class (produced by pack()).
- the output of the function can be
  - a single metric value, if the first argument is an instance of the gp class;
  - a single metric value or a vector of metric values with the length equal to the number of output dimensions, if the first argument is an instance of the dgp class;
  - a single metric value metric or a vector of metric values with the length equal to the number of emulators in the bundle, if the first argument is an instance of the bundle class.

If no customized function is provided, the built-in evaluation metric, RMSE, will be calculated. Defaults to NULL. See *Note* section below for further information.

`verb` a bool indicating if the trace information will be printed during the sequential design. Defaults to TRUE.

`autosave` a list that contains configuration settings for the automatic saving of the emulator:

- switch: a bool indicating whether to enable the automatic saving of the emulator during the sequential design. When set to TRUE, the emulator in the final iteration is always saved. Defaults to FALSE.
- directory: a string specifying the directory path where the emulators will be stored. Emulators will be stored in a sub-directory of directory named 'emulator-id'. Defaults to './check_points'.
- fname: a string representing the base name for the saved emulator files. Defaults to 'check_point'.
- freq: an integer indicating the frequency of automatic savings, measured in the number of iterations. Defaults to 5.
- overwrite: a bool value controlling the file saving behavior. When set to TRUE, each new automatic saving overwrites the previous one, keeping only the latest version. If FALSE, each automatic saving creates a new file, preserving all previous versions. Defaults to FALSE.

`new_wave` a bool indicating if the current execution of design() will create a new wave of sequential designs or add the sequential designs to the last existing wave. This argument is only used if there are waves existing in the emulator. By creating new waves, one can better visualize the performance of the sequential designs in different executions of design() in draw() and can specify a different evaluation frequency in freq. However, it can be beneficiary to turn this option off to restrict a large number of waves to be visualized in draw() that could run out of colors. Defaults to TRUE.

`M_val` an integer that gives the size of the conditioning set for the Vecchia approximation in emulator validations. This argument is only used if the emulator object was constructed under the Vecchia approximation. Defaults to 50.

`cores` an integer that gives the number of processes to be used for emulator validations. If set to NULL, the number of processes is set to max physical cores available %/% 2. Defaults to 1. This argument is only used if eval = NULL.

`...` any arguments (with names different from those of arguments used in `design()`) that are used by `f`, `method`, and `eval` can be passed here. `design()` will pass relevant arguments to `f`, `method`, and `eval` based on the names of additional arguments provided.

`train_N` the number of training iterations to be used to re-fit the DGP emulator at each step of the sequential design:

- If `train_N` is an integer, then at each step the DGP emulator will be re-fitted (based on the frequency of re-fit specified in `freq`) with `train_N` iterations.
- If `train_N` is a vector, then its size must be `N` even the re-fit frequency specified in `freq` is not one.
- If `train_N` is `NULL`, then at each step the DGP emulator will be re-fitted (based on the frequency of re-fit specified in `freq`) with `100` iterations if the DGP emulator was constructed without the Vecchia approximation, and with `50` iterations if Vecchia approximation was used.

Defaults to `NULL`.

`refit_cores` the number of processes to be used to re-fit GP components (in the same layer of a DGP emulator) at each M-step during the re-fitting. If set to `NULL`, the number of processes is set to `(max physical cores available - 1)` if the DGP emulator was constructed without the Vecchia approximation. Otherwise, the number of processes is set to `max physical cores available %/% 2`. Only use multiple processes when there is a large number of GP components in different layers and optimization of GP components is computationally expensive. Defaults to `1`.

`pruning` a bool indicating if dynamic pruning of DGP structures will be implemented during the sequential design after the total number of design points exceeds `min_size` in `control`. The argument is only applicable to DGP emulators (i.e., `object` is an instance of `dgp` class) produced by `dgp()` with `struc = NULL`. Defaults to `TRUE`.

`control` a list that can supply any of the following components to control the dynamic pruning of the DGP emulator:

- `min_size`, the minimum number of design points required to trigger the dynamic pruning. Defaults to 10 times of the input dimensions.
- `threshold`, the R2 value above which a GP node is considered redundant. Defaults to `0.97`.
- `nexceed`, the minimum number of consecutive iterations that the R2 value of a GP node must exceed `threshold` to trigger the removal of that node from the DGP structure. Defaults to `3`.

The argument is only used when `pruning = TRUE`.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

An updated `object` is returned with a slot called `design` that contains:

- *S* slots, named `wave1, wave2, ..., waveS`, that contain information of *S* waves of sequential designs that have been applied to the emulator. Each slot contains the following elements:
  - `N`, an integer that gives the numbers of steps implemented in the corresponding wave;
  - `rmse`, a matrix that gives the RMSEs of emulators constructed during the corresponding wave, if `eval = NULL`;
  - `metric`, a matrix that gives the customized evaluating metric values of emulators constructed during the corresponding wave, if a customized function is supplied to `eval`;
  - `freq`, an integer that gives the frequency that the emulator validations are implemented during the corresponding wave.
  - `enrichment`, a vector of size `N` that gives the number of new design points added after each step of the sequential design (if `object` is an instance of the `gp` or `dgp` class), or a matrix that gives the number of new design points added to emulators in a bundle after each step of the sequential design (if `object` is an instance of the `bundle` class).

  If `target` is not `NULL`, the following additional elements are also included:
  - `target`, the target RMSE(s) to stop the sequential design.
  - `reached`, a bool (if `object` is an instance of the `gp` or `dgp` class) or a vector of bools (if `object` is an instance of the `bundle` class) that indicate if the target RMSEs are reached at the end of the sequential design.
- a slot called `type` that gives the type of validations:
  - either LOO ('loo') or OOS ('oos') if `eval = NULL`. See `validate()` for more information about LOO and OOS.
  - 'customized' if a customized R function is provided to `eval`.
- two slots called `x_test` and `y_test` that contain the data points for the OOS validation if the `type` slot is 'oos'.
- If `y_cand = NULL` and there are `NA`s returned from the supplied `f` during the sequential design, a slot called `exclusion` is included that records the located design positions that produced `NA`s via `f`. The sequential design will use this information to avoid re-visiting the same locations (if `x_cand` is supplied) or their neighborhoods (if `x_cand` is `NULL`) in later runs of `design()`.

See *Note* section below for further information.

Note

- The validation of an emulator is forced after the final step of a sequential design even `N` is not multiples of the second element in `freq`.
- Any `loo` or `oos` slot that already exists in `object` will be cleaned, and a new slot called `loo` or `oos` will be created in the returned object depending on whether `x_test` and `y_test` are provided. The new slot gives the validation information of the emulator constructed in the final step of the sequential design. See `validate()` for more information about the slots `loo` and `oos`.
- If `object` has previously been used by `design()` for sequential designs, the information of the current wave of the sequential design will replace those of old waves and be contained in the returned object, unless
  - the validation type (LOO or OOS depending on whether `x_test` and `y_test` are supplied or not) of the current wave of the sequential design is the same as the validation types

(shown in the type of the design slot of object) in previous waves, and if the validation type is OOS, x_test and y_test in the current wave must also be identical to those in the previous waves;
  - both the current and previous waves of the sequential design supply customized evaluation functions to eval. Users need to ensure the customized evaluation functions are consistent among different waves. Otherwise, the trace plot of RMSEs produced by draw() will show values of different evaluation metrics in different waves.

  In above two cases, the information of the current wave of the sequential design will be added to the design slot of the returned object under the name waveS.
- If object is an instance of the gp class and eval = NULL, the matrix in the rmse slot is single-columned. If object is an instance of the dgp or bundle class and eval = NULL, the matrix in the rmse slot can have multiple columns that correspond to different output dimensions or different emulators in the bundle.
- If object is an instance of the gp class and eval = NULL, target needs to be a single value giving the RMSE threshold. If object is an instance of the dgp or bundle class and eval = NULL, target can be a vector of values that gives the RMSE thresholds for different output dimensions or different emulators. If a single value is provided, it will be used as the RMSE threshold for all output dimensions (if object is an instance of the dgp) or all emulators (if object is an instance of the bundle). If a customized function is supplied to eval, the user needs to ensure that the length of target is equal to that of the output from eval.
- When defining f, it is important to ensure that:
  - the column order of the first argument of f is consistent with the training input used for the emulator;
  - the column order of the output matrix of f is consistent with the order of emulator output dimensions (if object is an instance of the dgp class), or the order of emulators placed in object (if object is an instance of the bundle class).
- The output matrix produced by f may include NAs. This is especially beneficial as it allows the sequential design process to continue without interruption, even if errors or NA outputs are encountered from f at certain input locations identified by the sequential designs. Users should ensure to handle any errors within f by appropriately returning NAs.
- When defining eval, the output metric needs to be positive if draw() is used with log = T. And one needs to ensure that a lower metric value indicates a better emulation performance if target is set.
- Any R vector detected in x_test and y_test will be treated as a column vector and automatically converted into a single-column R matrix. Thus, if x_test or y_test is a single testing data point with multiple dimensions, it must be given as a matrix.

Examples

```
## Not run:

# load packages and the Python env
library(lhs)
library(dgpsi)

# construct a 2D non-stationary function that takes a matrix as the input
f <- function(x) {
```

```
  sin(1/((0.7*x[,1,drop=F]+0.3)*(0.7*x[,2,drop=F]+0.3)))
}

# generate the initial design
X <- maximinLHS(5,2)
Y <- f(X)

# generate the validation data
validate_x <- maximinLHS(30,2)
validate_y <- f(validate_x)

# training a 2-layered DGP emulator with the initial design
m <- dgp(X, Y)

# specify the ranges of the input dimensions
lim_1 <- c(0, 1)
lim_2 <- c(0, 1)
lim <- rbind(lim_1, lim_2)

# 1st wave of the sequential design with 10 steps
m <- design(m, N=10, limits = lim, f = f, x_test = validate_x, y_test = validate_y)

# 2nd wave of the sequential design with 10 steps
m <- design(m, N=10, limits = lim, f = f, x_test = validate_x, y_test = validate_y)

# 3rd wave of the sequential design with 10 steps
m <- design(m, N=10, limits = lim, f = f, x_test = validate_x, y_test = validate_y)

# draw the design created by the sequential design
draw(m,'design')

# inspect the trace of RMSEs during the sequential design
draw(m,'rmse')

# reduce the number of imputations for faster OOS
m_faster <- set_imp(m, 5)

# plot the OOS validation with the faster DGP emulator
plot(m_faster, x_test = validate_x, y_test = validate_y)

## End(Not run)
```

---

dgp *Deep Gaussian process emulator construction*

Description

This function builds and trains a DGP emulator.

Usage

```
dgp(
  X,
  Y,
  struc = NULL,
  depth = 2,
  node = ncol(X),
  name = "sexp",
  lengthscale = 1,
  bounds = NULL,
  prior = "ga",
  share = TRUE,
  nugget_est = FALSE,
  nugget = ifelse(all(nugget_est), 0.01, 1e-06),
  scale_est = TRUE,
  scale = 1,
  connect = TRUE,
  likelihood = NULL,
  training = TRUE,
  verb = TRUE,
  check_rep = TRUE,
  vecchia = FALSE,
  M = 25,
  ord = NULL,
  N = ifelse(vecchia, 200, 500),
  cores = 1,
  blocked_gibbs = TRUE,
  ess_burn = 10,
  burnin = NULL,
  B = 10,
  internal_input_idx = NULL,
  linked_idx = NULL,
  id = NULL
)
```

Arguments

X	a matrix where each row is an input training data point and each column is an input dimension.
Y	a matrix containing observed training output data. The matrix has its rows being output data points and columns being output dimensions. When likelihood (see below) is not NULL, Y must be a matrix with only one column.
struc	a list that specifies a user-defined DGP structure. It should contain L (the number of DGP layers) sub-lists, each of which represents a layer and contains a number of GP nodes (defined by kernel()) in the corresponding layer. The final layer of the DGP structure (i.e., the final sub-list in struc) can be a likelihood layer that contains a likelihood function (e.g., Poisson()). When struc = NULL, the DGP structure is automatically generated and can be checked by applying

summary() to the output from dgp() with training = FALSE. If this argument is used (i.e., user provides a customized DGP structure), arguments depth, node, name, lengthscale, bounds, prior, share, nugget_est, nugget, scale_est, scale, connect, likelihood, and internal_input_idx will NOT be used. Defaults to NULL.

depth — number of layers (including the likelihood layer) for a DGP structure. depth must be at least 2. Defaults to 2. This argument is only used when struc = NULL.

node — number of GP nodes in each layer (except for the final layer or the layer feeding the likelihood node) of the DGP. Defaults to ncol(X). This argument is only used when struc = NULL.

name — a character or a vector of characters that indicates the kernel functions (either "sexp" for squared exponential kernel or "matern2.5" for Matérn-2.5 kernel) used in the DGP emulator: 1. if a single character is supplied, the corresponding kernel function will be used for all GP nodes in the DGP hierarchy. 2. if a vector of characters is supplied, each character of the vector specifies the kernel function that will be applied to all GP nodes in the corresponding layer.

Defaults to "sexp". This argument is only used when struc = NULL.

lengthscale — initial lengthscales for GP nodes in the DGP emulator. It can be a single numeric value or a vector:

1. if it is a single numeric value, the value will be applied as the initial lengthscales for all GP nodes in the DGP hierarchy.
2. if it is a vector, each element of the vector specifies the initial lengthscales that will be applied to all GP nodes in the corresponding layer. The vector should have a length of depth if likelihood = NULL or a length of depth - 1 if likelihood is not NULL.

Defaults to a numeric value of 1.0. This argument is only used when struc = NULL.

bounds — the lower and upper bounds of lengthscales in GP nodes. It can be a vector or a matrix:

1. if it is a vector, the lower bound (the first element of the vector) and upper bound (the second element of the vector) will be applied to lengthscales for all GP nodes in the DGP hierarchy.
2. if it is a matrix, each row of the matrix specifies the lower and upper bounds of lengthscales for all GP nodes in the corresponding layer. The matrix should have its row number equal to depth if likelihood = NULL or to depth - 1 if likelihood is not NULL.

Defaults to NULL where no bounds are specified for the lengthscales. This argument is only used when struc = NULL.

prior — prior to be used for Maximum a Posterior for lengthscales and nuggets of all GP nodes in the DGP hierarchy:

- gamma prior ("ga"),
- inverse gamma prior ("inv_ga"), or
- jointly robust prior ("ref").

Defaults to "ga". This argument is only used when struc = NULL.

share — a bool indicating if all input dimensions of a GP node share a common length-scale. Defaults to TRUE. This argument is only used when struc = NULL.

nugget_est — a bool or a bool vector that indicates if the nuggets of GP nodes (if any) in the final layer are to be estimated. If a single bool is provided, it will be applied to all GP nodes (if any) in the final layer. If a bool vector (which must have a length of ncol(Y)) is provided, each bool element in the vector will be applied to the corresponding GP node (if any) in the final layer. The value of a bool has following effects:

- FALSE: the nugget of the corresponding GP in the final layer is fixed to the corresponding value defined in nugget (see below).
- TRUE: the nugget of the corresponding GP in the final layer will be estimated with the initial value given by the correspondence in nugget (see below).

Defaults to FALSE. This argument is only used when struc = NULL.

nugget — the initial nugget value(s) of GP nodes (if any) in each layer:

1. if it is a single numeric value, the value will be applied as the initial nugget for all GP nodes in the DGP hierarchy.
2. if it is a vector, each element of the vector specifies the initial nugget that will be applied to all GP nodes in the corresponding layer. The vector should have a length of depth if likelihood = NULL or a length of depth - 1 if likelihood is not NULL.

Set nugget to a small value and the bools in nugget_est to FASLE for deterministic emulations where the emulator interpolates the training data points. Set nugget to a reasonable larger value and the bools in nugget_est to TRUE for stochastic emulations where the computer model outputs are assumed to follow a homogeneous Gaussian distribution. Defaults to 1e-6 if nugget_est = FALSE and 0.01 if nugget_est = TRUE. This argument is only used when struc = NULL.

scale_est — a bool or a bool vector that indicates if variance of GP nodes (if any) in the final layer are to be estimated. If a single bool is provided, it will be applied to all GP nodes (if any) in the final layer. If a bool vector (which must have a length of ncol(Y)) is provided, each bool element in the vector will be applied to the corresponding GP node (if any) in the final layer. The value of a bool has following effects:

- FALSE: the variance of the corresponding GP in the final layer is fixed to the corresponding value defined in scale (see below).
- TRUE: the variance of the corresponding GP in the final layer will be estimated with the initial value given by the correspondence in scale (see below).

Defaults to TRUE. This argument is only used when struc = NULL.

scale — the initial variance value(s) of GP nodes (if any) in the final layer. If it is a single numeric value, it will be applied to all GP nodes (if any) in the final layer. If it is a vector (which must have a length of ncol(Y)), each numeric in the vector will be applied to the corresponding GP node (if any) in the final layer. Defaults to 1. This argument is only used when struc = NULL.

connect — a bool indicating whether to implement global input connection to the DGP structure. Setting it to FALSE may produce a better emulator in some cases at the cost of slower training. Defaults to TRUE. This argument is only used when struc = NULL.

likelihood — the likelihood type of a DGP emulator:

1. NULL: no likelihood layer is included in the emulator.
2. "Hetero": a heteroskedastic Gaussian likelihood layer is added for stochastic emulation where the computer model outputs are assumed to follow a heteroskedastic Gaussian distribution (i.e., the computer model outputs have varying noises).
3. "Poisson": a Poisson likelihood layer is added for stochastic emulation where the computer model outputs are assumed to a Poisson distribution.
4. "NegBin": a negative Binomial likelihood layer is added for stochastic emulation where the computer model outputs are assumed to follow a negative Binomial distribution.

When likelihood is not NULL, the value of nugget_est is overridden by FALSE. Defaults to NULL. This argument is only used when struc = NULL.

training — a bool indicating if the initialized DGP emulator will be trained. When set to FALSE, dgp() returns an untrained DGP emulator, to which one can apply summary() to inspect its specifications (especially when a customized struc is provided) or apply predict() to check its emulation performance before the training. Defaults to TRUE.

verb — a bool indicating if the trace information on DGP emulator construction and training will be printed during the function execution. Defaults to TRUE.

check_rep — a bool indicating whether to check the repetitions in the dataset, i.e., if one input position has multiple outputs. Defaults to TRUE.

vecchia — a bool indicating whether to use Vecchia approximation for large-scale DGP emulator construction and prediction. Defaults to FALSE.

M — the size of the conditioning set for the Vecchia approximation in the DGP emulator training. Defaults to 25.

ord — an R function that returns the ordering of the input to each GP node contained in the DGP emulator for the Vecchia approximation. The function must satisfy the following basic rules:

- the first argument represents the input to a GP node scaled by its lengthscales.
- the output of the function is a vector of indices that gives the ordering of the input to the GP node.

If ord = NULL, the default random ordering is used. Defaults to NULL.

N — number of iterations for the training. Defaults to 500 if vecchia = FALSE and 200 if vecchia = TRUE. This argument is only used when training = TRUE.

cores — the number of processes to be used to optimize GP components (in the same layer) at each M-step of the training. If set to NULL, the number of processes is set to (max physical cores available - 1) if vecchia = FALSE and max physical cores available %/% 2 if vecchia = TRUE. Only use multiple

processes when there is a large number of GP components in different layers and optimization of GP components is computationally expensive. Defaults to 1.

`blocked_gibbs` a bool indicating if the latent variables are imputed layer-wise using ESS-within-Blocked-Gibbs. ESS-within-Blocked-Gibbs would be faster and more efficient than ESS-within-Gibbs that imputes latent variables node-wise because it reduces the number of components to be sampled during the Gibbs, especially when there is a large number of GP nodes in layers due to higher input dimensions. Default to TRUE.

`ess_burn` number of burnin steps for the ESS-within-Gibbs at each I-step of the training. Defaults to 10. This argument is only used when `training = TRUE`.

`burnin` the number of training iterations to be discarded for point estimates of model parameters. Must be smaller than the training iterations N. If this is not specified, only the last 25% of iterations are used. Defaults to NULL. This argument is only used when `training = TRUE`.

`B` the number of imputations to produce the later predictions. Increase the value to account for more imputation uncertainties with slower predictions. Decrease the value for lower imputation uncertainties but faster predictions. Defaults to 10.

`internal_input_idx`

column indices of X that are generated by the linked emulators in the preceding layers. Set `internal_input_idx = NULL` if the DGP emulator is in the first layer of a system or all columns in X are generated by the linked emulators in the preceding layers. Defaults to NULL. This argument is only used when `struc = NULL`.

`linked_idx` either a vector or a list of vectors:

- If `linked_idx` is a vector, it gives indices of columns in the pooled output matrix (formed by column-combined outputs of all emulators in the feeding layer) that feed into the DGP emulator. The length of the vector shall equal to the length of `internal_input_idx` when `internal_input_idx` is not NULL. If the DGP emulator is in the first layer of a linked emulator system, the vector gives the column indices of the global input (formed by column-combining all input matrices of emulators in the first layer) that the DGP emulator will use. If the DGP emulator is to be used in both the first and subsequent layers, one should initially set `linked_idx` to the appropriate values for the situation where the emulator is not in the first layer. Then, use the function `set_linked_idx()` to reset the linking information when the emulator is in the first layer.
- When the DGP emulator is not in the first layer of a linked emulator system, `linked_idx` can be a list that gives the information on connections between the DGP emulator and emulators in all preceding layers. The length of the list should equal to the number of layers before the DGP emulator. Each element of the list is a vector that gives indices of columns in the pooled output matrix (formed by column-combined outputs of all emulators) in the corresponding layer that feed into the DGP emulator. If the DGP emulator has no connections to any emulator in a certain layer, set NULL in the corresponding position of the list. The order of input dimensions in

`X[,internal_input_idx]` should be consistent with `linked_idx`. For example, a DGP emulator in the 4th-layer that is fed by the output dimension 2 and 4 of emulators in layer 2 and all output dimension 1 to 3 of emulators in layer 3 should have `linked_idx = list( NULL, c(2,4), c(1,2,3) )`. In addition, the first and second columns of `X[,internal_input_idx]` should correspond to the output dimensions 2 and 4 from layer 2, and the third to fifth columns of `X[,internal_input_idx]` should correspond to the output dimensions 1 to 3 from layer 3.

Set `linked_idx = NULL` if the DGP emulator will not be used for linked emulations. However, if this is no longer the case, one can use `set_linked_idx()` to add linking information to the DGP emulator. Defaults to `NULL`.

`id` an ID to be assigned to the DGP emulator. If an ID is not provided (i.e., `id = NULL`), a UUID (Universally Unique Identifier) will be automatically generated and assigned to the emulator. Default to `NULL`.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/ and learn how to customize a DGP structure.

Value

An S3 class named dgp that contains five slots:

- `id`: A number or character string assigned through the `id` argument.
- `data`: a list that contains two elements: `X` and `Y` which are the training input and output data respectively.
- `specs`: a list that contains
  1. *L* (i.e., the number of layers in the DGP hierarchy) sub-lists named `layer1, layer2,..., layerL`. Each sub-list contains *D* (i.e., the number of GP/likelihood nodes in the corresponding layer) sub-lists named `node1, node2,..., nodeD`. If a sub-list corresponds to a likelihood node, it contains one element called `type` that gives the name (`Hetero`, `Poisson`, or `NegBin`) of the likelihood node. If a sub-list corresponds to a GP node, it contains four elements:
     - `kernel`: the type of the kernel function used for the GP node.
     - `lengthscales`: a vector of lengthscales in the kernel function.
     - `scale`: the variance value in the kernel function.
     - `nugget`: the nugget value in the kernel function.
  2. `internal_dims`: the column indices of `X` that correspond to the linked emulators in the preceding layers of a linked system.
  3. `external_dims`: the column indices of `X` that correspond to global inputs to the linked system of emulators. It is shown as `FALSE` if `internal_input_idx = NULL`.
  4. `linked_idx`: the value passed to argument `linked_idx`. It is shown as `FALSE` if the argument `linked_idx` is `NULL`.
  5. `seed`: the random seed generated to produce the imputations. This information is stored for the reproducibility when the DGP emulator (that was saved by `write()` with the light option `light = TRUE`) is loaded back to R by `read()`.

6. B: the number of imputations used to generate the emulator.
7. vecchia: whether the Vecchia approximation is used for the GP emulator training.
8. M: the size of the conditioning set for the Vecchia approximation in the DGP emulator training.

internal_dims and external_dims are generated only when struc = NULL. M is generated only when vecchia = TRUE.

- constructor_obj: a 'python' object that stores the information of the constructed DGP emulator.
- container_obj: a 'python' object that stores the information for the linked emulation.
- emulator_obj: a 'python' object that stores the information for the predictions from the DGP emulator.

The returned dgp object can be used by

- predict() for DGP predictions.
- continue() for additional DGP training iterations.
- validate() for LOO and OOS validations.
- plot() for validation plots.
- lgp() for linked (D)GP emulator constructions.
- window() for model parameter trimming.
- summary() to summarize the trained DGP emulator.
- write() to save the DGP emulator to a .pkl file.
- set_imp() to change the number of imputations.
- set_linked_idx() to add the linking information to the DGP emulator for linked emulations.
- design() for sequential designs.
- update() to update the DGP emulator with new inputs and outputs.
- alm(), mice(), pei(), and vigf() to locate next design points.

Note

Any R vector detected in X and Y will be treated as a column vector and automatically converted into a single-column R matrix. Thus, if X is a single data point with multiple dimensions, it must be given as a matrix.

Examples

```
## Not run:

# load the package and the Python env
library(dgpsi)

# construct a step function
f <- function(x) {
  if (x < 0.5) return(-1)
  if (x >= 0.5) return(1)
```

```
  }

# generate training data
X <- seq(0, 1, length = 10)
Y <- sapply(X, f)

# set a random seed
set_seed(999)

# training a DGP emulator
m <- dgp(X, Y)

# continue for further training iterations
m <- continue(m)

# summarizing
summary(m)

# trace plot
trace_plot(m)

# trim the traces of model parameters
m <- window(m, 800)

# LOO cross validation
m <- validate(m)
plot(m)

# prediction
test_x <- seq(0, 1, length = 200)
m <- predict(m, x = test_x)

# OOS validation
validate_x <- sample(test_x, 10)
validate_y <- sapply(validate_x, f)
plot(m, validate_x, validate_y)

# write and read the constructed emulator
write(m, 'step_dgp')
m <- read('step_dgp')

## End(Not run)
```

---

draw *Validation plots of a sequential design*

Description

This function draws validation plots of the sequential design of a (D)GP emulator or a bundle of (D)GP emulators.

Usage

```
draw(object, ...)

## S3 method for class 'gp'
draw(object, type = "rmse", log = FALSE, ...)

## S3 method for class 'dgp'
draw(object, type = "rmse", log = FALSE, ...)

## S3 method for class 'bundle'
draw(object, emulator = 1, type = "rmse", log = FALSE, ...)
```

Arguments

object	can be one of the following emulator classes: • the S3 class gp. • the S3 class dgp. • the S3 class bundle.
...	N/A.
type	either "rmse", for the trace plot of RMSEs or customized evaluating metrics of emulators constructed during the sequential designs, or "design", for visualizations of input designs created by the sequential design procedure. Defaults to "rmse".
log	a bool that indicates whether to plot RMSEs or customized evaluating metrics in log-scale if type = "rmse". Defaults to FALSE.
emulator	the index of the emulator packed in object if object is an instance of the bundle class.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

A patchwork object.

Note

If a customized evaluating function is provided to design() and the function returns a single evaluating metric value when object is an instance of the bundle class, the value of emulator has no effects on the plot when type = "rmse".

Examples

```
## Not run:

# See design() for an example.
```

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

---

get_thread_num *Get the number of threads*

Description

This function gets the number of threads used for parallel computations involved in the package.

Usage

```
get_thread_num()
```

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

the number of threads.

---

gp *Gaussian process emulator construction*

Description

This function builds and trains a GP emulator.

Usage

```
gp(
  X,
  Y,
  struc = NULL,
  name = "sexp",
  lengthscale = rep(0.1, ncol(X)),
  bounds = NULL,
  prior = "ref",
  nugget_est = FALSE,
  nugget = ifelse(nugget_est, 0.01, 1e-08),
  scale_est = TRUE,
  scale = 1,
  training = TRUE,
  verb = TRUE,
  vecchia = FALSE,
```

```
  M = 25,
  ord = NULL,
  internal_input_idx = NULL,
  linked_idx = NULL,
  id = NULL
)
```

Arguments

X	a matrix where each row is an input data point and each column is an input dimension.
Y	a matrix with only one column and each row being an output data point.
struc	an object produced by kernel() that gives a user-defined GP specifications. When struc = NULL, the GP specifications are automatically generated using information provided in name, lengthscale, nugget_est, nugget, scale_est, scale,and internal_input_idx. Defaults to NULL.
name	kernel function to be used. Either "sexp" for squared exponential kernel or "matern2.5" for Matérn-2.5 kernel. Defaults to "sexp". This argument is only used when struc = NULL.
lengthscale	initial values of lengthscales in the kernel function. It can be a single numeric value or a vector: • if it is a single numeric value, it is assumed that kernel functions across input dimensions share the same lengthscale; • if it is a vector (which must have a length of ncol(X)), it is assumed that kernel functions across input dimensions have different lengthscales. Defaults to a vector of 0.1. This argument is only used when struc = NULL.
bounds	the lower and upper bounds of lengthscales in the kernel function. It is a vector of length two where the first element is the lower bound and the second element is the upper bound. The bounds will be applied to all lengthscales in the kernel function. Defaults to NULL where no bounds are specified for the lengthscales. This argument is only used when struc = NULL.
prior	prior to be used for Maximum a Posterior for lengthscales and nugget of the GP: gamma prior ("ga"), inverse gamma prior ("inv_ga"), or jointly robust prior ("ref"). Defaults to "ref". This argument is only used when struc = NULL. See the reference below for the jointly robust prior.
nugget_est	a bool indicating if the nugget term is to be estimated: 1. FALSE: the nugget term is fixed to nugget. 2. TRUE: the nugget term will be estimated. Defaults to FALSE. This argument is only used when struc = NULL.
nugget	the initial nugget value. If nugget_est = FALSE, the assigned value is fixed during the training. Set nugget to a small value (e.g., 1e-8) and the corresponding bool in nugget_est to FASLE for deterministic emulations where the emulator interpolates the training data points. Set nugget to a reasonable larger value and the corresponding bool in nugget_est to TRUE for stochastic emulations where the computer model outputs are assumed to follow a homogeneous Gaussian

distribution. Defaults to `1e-8` if `nugget_est = FALSE` and `0.01` if `nugget_est = TRUE`. This argument is only used when `struc = NULL`.

`scale_est` a bool indicating if the variance is to be estimated:

1. `FALSE`: the variance is fixed to `scale`.
2. `TRUE`: the variance term will be estimated.

Defaults to `TRUE`. This argument is only used when `struc = NULL`.

`scale` the initial variance value. If `scale_est = FALSE`, the assigned value is fixed during the training. Defaults to `1`. This argument is only used when `struc = NULL`.

`training` a bool indicating if the initialized GP emulator will be trained. When set to `FALSE`, `gp()` returns an untrained GP emulator, to which one can apply `summary()` to inspect its specifications (especially when a customized `struc` is provided) or apply `predict()` to check its emulation performance before the training. Defaults to `TRUE`.

`verb` a bool indicating if the trace information on GP emulator construction and training will be printed during the function execution. Defaults to `TRUE`.

`vecchia` a bool indicating whether to use Vecchia approximation for large-scale GP emulator construction and prediction. Defaults to `FALSE`. The Vecchia approximation implemented for the GP emulation largely follows Katzfuss et al. (2022). See reference below.

`M` the size of the conditioning set for the Vecchia approximation in the GP emulator training. Defaults to `25`.

`ord` an R function that returns the ordering of the input to the GP emulator for the Vecchia approximation. The function must satisfy the following basic rules:

- the first argument represents the input scaled by the lengthscales.
- the output of the function is a vector of indices that gives the ordering of the input to the GP emulator.

If `ord = NULL`, the default random ordering is used. Defaults to `NULL`.

`internal_input_idx`
the column indices of `X` that are generated by the linked emulators in the preceding layers. Set `internal_input_idx = NULL` if the GP emulator is in the first layer of a system or all columns in `X` are generated by the linked emulators in the preceding layers. Defaults to `NULL`. This argument is only used when `struc = NULL`.

`linked_idx` either a vector or a list of vectors:

- If `linked_idx` is a vector, it gives indices of columns in the pooled output matrix (formed by column-combined outputs of all emulators in the feeding layer) that feed into the GP emulator. The length of the vector shall equal to the length of `internal_input_idx` when `internal_input_idx` is not `NULL`. If the GP emulator is in the first layer of a linked emulator system, the vector gives the column indices of the global input (formed by column-combining all input matrices of emulators in the first layer) that the GP emulator will use. If the GP emulator is to be used in both the first and subsequent layers, one should initially set `linked_idx` to the appropriate

values for the situation where the emulator is not in the first layer. Then, use the function set_linked_idx() to reset the linking information when the emulator is in the first layer.

- When the GP emulator is not in the first layer of a linked emulator system, linked_idx can be a list that gives the information on connections between the GP emulator and emulators in all preceding layers. The length of the list should equal to the number of layers before the GP emulator. Each element of the list is a vector that gives indices of columns in the pooled output matrix (formed by column-combined outputs of all emulators) in the corresponding layer that feed into the GP emulator. If the GP emulator has no connections to any emulator in a certain layer, set NULL in the corresponding position of the list. The order of input dimensions in X[,internal_input_idx] should be consistent with linked_idx. For example, a GP emulator in the second layer that is fed by the output dimension 1 and 3 of emulators in layer 1 should have linked_idx = list(c(1,3)). In addition, the first and second columns of X[,internal_input_idx] should correspond to the output dimensions 1 and 3 from layer 1.

Set linked_idx = NULL if the GP emulator will not be used for linked emulations. However, if this is no longer the case, one can use set_linked_idx() to add linking information to the GP emulator. Defaults to NULL.

id — an ID to be assigned to the GP emulator. If an ID is not provided (i.e., id = NULL), a UUID (Universally Unique Identifier) will be automatically generated and assigned to the emulator. Default to NULL.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

An S3 class named gp that contains five slots:

- id: A number or character string assigned through the id argument.
- data: a list that contains two elements: X and Y which are the training input and output data respectively.
- specs: a list that contains seven elements:
  1. kernel: the type of the kernel function used. Either "sexp" for squared exponential kernel or "matern2.5" for Matérn-2.5 kernel.
  2. lengthscales: a vector of lengthscales in the kernel function.
  3. scale: the variance value in the kernel function.
  4. nugget: the nugget value in the kernel function.
  5. internal_dims: the column indices of X that correspond to the linked emulators in the preceding layers of a linked system.
  6. external_dims: the column indices of X that correspond to global inputs to the linked system of emulators. It is shown as FALSE if internal_input_idx = NULL.
  7. linked_idx: the value passed to argument linked_idx. It is shown as FALSE if the argument linked_idx is NULL.

8. `vecchia`: whether the Vecchia approximation is used for the GP emulator training.
9. `M`: the size of the conditioning set for the Vecchia approximation in the GP emulator training.

`internal_dims` and `external_dims` are generated only when `struc = NULL`.

- `constructor_obj`: a 'python' object that stores the information of the constructed GP emulator.
- `container_obj`: a 'python' object that stores the information for the linked emulation.
- `emulator_obj`: a 'python' object that stores the information for the predictions from the GP emulator.

The returned gp object can be used by

- `predict()` for GP predictions.
- `validate()` for LOO and OOS validations.
- `plot()` for validation plots.
- `lgp()` for linked (D)GP emulator constructions.
- `summary()` to summarize the trained GP emulator.
- `write()` to save the GP emulator to a `.pkl` file.
- `set_linked_idx()` to add the linking information to the GP emulator for linked emulations.
- `design()` for sequential designs.
- `update()` to update the GP emulator with new inputs and outputs.
- `alm()`, `mice()`, `pei()`, and `vigf()` to locate next design points.

Note

Any R vector detected in `X` and `Y` will be treated as a column vector and automatically converted into a single-column R matrix. Thus, if `X` is a single data point with multiple dimensions, it must be given as a matrix.

References

- Gu, M. (2019). Jointly robust prior for Gaussian stochastic process in emulation, calibration and variable selection. *Bayesian Analysis*, **14(3)**, 857-885.
- Katzfuss, M., Guinness, J., & Lawrence, E. (2022). Scaled Vecchia approximation for fast computer-model emulation. *SIAM/ASA Journal on Uncertainty Quantification*, **10(2)**, 537-554.

Examples

```
## Not run:
# load the package and the Python env
library(dgpsi)

# construct a step function
f <- function(x) {
   if (x < 0.5) return(-1)
```

```
   if (x >= 0.5) return(1)
  }

# generate training data
X <- seq(0, 1, length = 10)
Y <- sapply(X, f)

# training
m <- gp(X, Y)

# summarizing
summary(m)

# LOO cross validation
m <- validate(m)
plot(m)

# prediction
test_x <- seq(0, 1, length = 200)
m <- predict(m, x = test_x)

# OOS validation
validate_x <- sample(test_x, 10)
validate_y <- sapply(validate_x, f)
plot(m, validate_x, validate_y)

# write and read the constructed emulator
write(m, 'step_gp')
m <- read('step_gp')

## End(Not run)
```

`Hetero` *Initialize a heteroskedastic Gaussian likelihood node*

Description

This function constructs a likelihood object to represent a heteroskedastic Gaussian likelihood node.

Usage

```
Hetero(input_dim = NULL)
```

Arguments

`input_dim`	a vector of length two that contains the indices of two GP nodes in the feeding layer whose outputs feed into this likelihood node. When set to `NULL`, all outputs from GP nodes in the feeding layer feed into this likelihood node, and in such a case one needs to ensure that only two GP nodes are specified in the feeding layer. Defaults to `NULL`.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

A 'python' object to represent a heteroskedastic Gaussian likelihood node.

Note

The heteroskedastic Gaussian likelihood node can only be linked to two feeding GP nodes.

Examples

```
## Not run:

# Check https://mingdeyu.github.io/dgpsi-R/ for examples
# on how to customize DGP structures using Hetero().

## End(Not run)
```

---

init_py *'python' environment initialization*

Description

This function initializes the 'python' environment for the package.

Usage

```
init_py(
  py_ver = NULL,
  dgpsi_ver = NULL,
  reinstall = FALSE,
  uninstall = FALSE,
  verb = TRUE
)
```

Arguments

py_ver	a string that gives the 'python' version to be installed. If py_ver = NULL, the default 'python' version '3.9.13' will be installed.
dgpsi_ver	a string that gives the 'python' version of 'dgpsi' to be used. If dgpsi_ver = NULL, • the latest 'python' version of 'dgpsi' will be used, if the package is installed from CRAN; • the development 'python' version of 'dgpsi' will be used, if the package is installed from GitHub.

reinstall	a bool that indicates whether to reinstall the 'python' version of 'dgpsi' specified in dgpsi_ver if it has already been installed. This argument is useful when the development version of the R package is installed and one may want to regularly update the development 'python' version of 'dgpsi'. Defaults to FALSE.
uninstall	a bool that indicates whether to uninstall the 'python' version of 'dgpsi' specified in dgpsi_ver if it has already been installed. This argument is useful when the 'python' environment is corrupted and one wants to completely uninstall and reinstall it. Defaults to FALSE.
verb	a bool indicating if the trace information will be printed during the function execution. Defaults to TRUE.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

No return value, called to install required 'python' environment.

Examples

```
## Not run:

# See gp(), dgp(), or lgp() for an example.

## End(Not run)
```

---

kernel *Initialize a Gaussian process node*

Description

This function constructs a kernel object to represent properties of a Gaussian process node.

Usage

```
kernel(
  length,
  scale = 1,
  nugget = 1e-06,
  name = "sexp",
  prior_name = "ga",
  prior_coef = NULL,
  bounds = NULL,
  nugget_est = FALSE,
  scale_est = FALSE,
```

```
  input_dim = NULL,
  connect = NULL
)
```

Arguments

length	a vector of lengthscales. The length of the vector equals to: 1. either one if the lengthscales in the kernel function are assumed same across input dimensions; or 2. the total number of input dimensions, which is the sum of the number of feeding GP nodes in the last layer (defined by the argument input_dim) and the number of connected global input dimensions (defined by the argument connect), if the lengthscales in the kernel function are assumed different across input dimensions.
scale	the variance of a GP node. Defaults to 1.
nugget	the nugget term of a GP node. Defaults to 1e-6.
name	kernel function to be used. Either "sexp" for squared exponential kernel or "matern2.5" for Matérn-2.5 kernel. Defaults to "sexp".
prior_name	prior options for the lengthscales and nugget term: gamma prior ("ga"), inverse gamma prior ("inv_ga"), or jointly robust prior ("ref") for the lengthscales and nugget term. Set NULL to disable the prior. Defaults to "ga".
prior_coef	a vector that contains the coefficients for different priors: • for the gamma prior, it is a vector of two values specifying the shape and rate parameters of the gamma distribution. Set to NULL for the default value c(1.6,0.3). • for the inverse gamma prior, it is a vector of two values specifying the shape and scale parameters of the inverse gamma distribution. Set to NULL for the default value c(1.6,0.3). • for the jointly robust prior, it is a vector of a single value specifying the a parameter in the prior. Set to NULL for the default value c(0.2). See the reference below for the jointly robust prior. Defaults to NULL.
bounds	a vector of length two that gives the lower bound (the first element of the vector) and the upper bound (the second element of the vector) of all lengthscales of the GP node. Defaults to NULL where no bounds are specified for the lengthscales.
nugget_est	set to TRUE to estimate the nugget term or to FALSE to fix the nugget term as specified by the argument nugget. If set to TRUE, the value set to the argument nugget is used as the initial value. Defaults to FALSE.
scale_est	set to TRUE to estimate the variance (i.e., scale) or to FALSE to fix the variance (i.e., scale) as specified by the argument scale. Defaults to FALSE.
input_dim	a vector that contains either 1. the indices of GP nodes in the feeding layer whose outputs feed into this GP node; or

2. the indices of global input dimensions that are linked to the outputs of some feeding emulators, if this GP node is in the first layer of a GP or DGP, which will be used for the linked emulation.

When set to NULL,

1. all outputs from the GP nodes in the feeding layer feed into this GP node; or
2. all global input dimensions feed into this GP node.

Defaults to NULL.

`connect` a vector that contains the indices of dimensions in the global input connecting to this GP node as additional input dimensions. When set to NULL, no global input connection is implemented. Defaults to NULL. When this GP node is in the first layer of a GP or DGP emulator, which will consequently be used for linked emulation, `connect` gives the indices of global input dimensions that are not connected to some feeding emulators. In such a case, set `input_dim` to a vector of indices of the remaining input dimensions that are connected to the feeding emulators.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

A 'python' object to represent a GP node.

References

Gu, M. (2019). Jointly robust prior for Gaussian stochastic process in emulation, calibration and variable selection. *Bayesian Analysis*, **14(3)**, 857-885.

Examples

```
## Not run:

# Check https://mingdeyu.github.io/dgpsi-R/ for examples
# on how to customize DGP structures using kernel().

## End(Not run)
```

---

lgp *Linked (D)GP emulator construction*

Description

This function constructs a linked (D)GP emulator.

Usage

```
lgp(struc, B = 10, id = NULL)
```

Arguments

struc	a list contains L (the number of layers in a systems of computer models) sub-lists, each of which represents a layer and contains (D)GP emulators (represented by instances of S3 class gp or dgp) of computer models. The sub-lists are placed in the list in the same order of the specified computer model system's hierarchy.
B	the number of imputations to produce the predictions. Increase the value to account for more imputation uncertainties. Decrease the value for lower imputation uncertainties but faster predictions. If the system consists only GP emulators, B is set to 1 automatically. Defaults to 10.
id	an ID to be assigned to the linked (D)GP emulator. If an ID is not provided (i.e., id = NULL), a UUID (Universally Unique Identifier) will be automatically generated and assigned to the emulator. Default to NULL.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

An S3 class named lgp that contains three slots:

- id: A number or character string assigned through the id argument.
- constructor_obj: a list of 'python' objects that stores the information of the constructed linked emulator.
- emulator_obj, a 'python' object that stores the information for predictions from the linked emulator.
- specs: a list that contains
  1. seed: the random seed generated to produce the imputations. This information is stored for the reproducibility when the linked (D)GP emulator (that was saved by write() with the light option light = TRUE) is loaded back to R by read().
  2. B: the number of imputations used to generate the linked (D)GP emulator.

The returned lgp object can be used by

- predict() for linked (D)GP predictions.
- validate() for the OOS validation.
- plot() for the validation plots.
- summary() to summarize the constructed linked (D)GP emulator.
- write() to save the linked (D)GP emulator to a .pkl file.

Examples

```
## Not run:

# load the package and the Python env
library(dgpsi)

# model 1
f1 <- function(x) {
  (sin(7.5*x)+1)/2
}
# model 2
f2 <- function(x) {
  2/3*sin(2*(2*x - 1))+4/3*exp(-30*(2*(2*x-1))^2)-1/3
}
# linked model
f12 <- function(x) {
  f2(f1(x))
}

# training data for Model 1
X1 <- seq(0, 1, length = 9)
Y1 <- sapply(X1, f1)
# training data for Model 2
X2 <- seq(0, 1, length = 13)
Y2 <- sapply(X2, f2)

# emulation of model 1
m1 <- gp(X1, Y1, name = "matern2.5", linked_idx = c(1))
# emulation of model 2
m2 <- dgp(X2, Y2, depth = 2, name = "matern2.5")
# assign linking information after the emulation construction
m2 <- set_linked_idx(m2, c(1))

# emulation of the linked model
struc <- combine(list(m1), list(m2))
m_link <- lgp(struc)

# summarizing
summary(m_link)

# prediction
test_x <- seq(0, 1, length = 300)
m_link <- predict(m_link, x = test_x)

# OOS validation
validate_x <- sample(test_x, 20)
validate_y <- sapply(validate_x, f12)
plot(m_link, validate_x, validate_y, style = 2)

# write and read the constructed linked emulator
write(m_link, 'linked_emulator')
m_link <- read('linked_emulator')
```

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

`mice` *Locate the next design point for a (D)GP emulator or a bundle of (D)GP emulators using MICE*

Description

This function searches from a candidate set to locate the next design point(s) to be added to a (D)GP emulator or a bundle of (D)GP emulators using the Mutual Information for Computer Experiments (MICE), see the reference below.

Usage

```
mice(object, x_cand, ...)

## S3 method for class 'gp'
mice(
  object,
  x_cand,
  batch_size = 1,
  M = 50,
  nugget_s = 1e-06,
  workers = 1,
  ...
)

## S3 method for class 'dgp'
mice(
  object,
  x_cand,
  batch_size = 1,
  M = 50,
  nugget_s = 1e-06,
  workers = 1,
  aggregate = NULL,
  ...
)

## S3 method for class 'bundle'
mice(
  object,
  x_cand,
  batch_size = 1,
  M = 50,
  nugget_s = 1e-06,
```

```
  workers = 1,
  aggregate = NULL,
  ...
)
```

Arguments

`object`	can be one of the following: • the S3 class `gp`. • the S3 class `dgp`. • the S3 class `bundle`.
`x_cand`	a matrix (with each row being a design point and column being an input dimension) that gives a candidate set from which the next design point(s) are determined. If `object` is an instance of the `bundle` class, `x_cand` could also be a list with the length equal to the number of emulators contained in the `object`. Each slot in `x_cand` is a matrix that gives a candidate set for each emulator included in the bundle. See *Note* section below for further information.
`...`	any arguments (with names different from those of arguments used in `mice()`) that are used by `aggregate` can be passed here.
`batch_size`	an integer that gives the number of design points to be chosen. Defaults to `1`.
`M`	the size of the conditioning set for the Vecchia approximation in the criterion calculation. This argument is only used if the emulator `object` was constructed under the Vecchia approximation. Defaults to `50`.
`nugget_s`	the value of the smoothing nugget term used by MICE. Defaults to `1e-6`.
`workers`	the number of processes to be used for the criterion calculation. If set to `NULL`, the number of processes is set to `max physical cores available %/% 2`. Defaults to `1`.
`aggregate`	an R function that aggregates scores of the MICE across different output dimensions (if `object` is an instance of the `dgp` class) or across different emulators (if `object` is an instance of the `bundle` class). The function should be specified in the following basic form: • the first argument is a matrix representing scores. The rows of the matrix correspond to different design points. The number of columns of the matrix equals to: – the emulator output dimension if `object` is an instance of the `dgp` class; or – the number of emulators contained in `object` if `object` is an instance of the `bundle` class. • the output should be a vector that gives aggregations of scores at different design points. Set to `NULL` to disable the aggregation. Defaults to `NULL`.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

- If `object` is an instance of the `gp` class, a vector is returned with the length equal to `batch_size`, giving the positions (i.e., row numbers) of next design points from `x_cand`.
- If `object` is an instance of the `dgp` class, a matrix is returned with row number equal to `batch_size` and column number equal to one (if `aggregate` is not `NULL`) or the output dimension (if `aggregate` is `NULL`), giving positions (i.e., row numbers) of next design points from `x_cand` to be added to the DGP emulator across different outputs. If `object` is a DGP emulator with either `Hetero` or `NegBin` likelihood layer, the returned matrix has two columns with the first column giving positions of next design points from `x_cand` that correspond to the mean parameter of the normal or negative Binomial distribution, and the second column giving positions of next design points from `x_cand` that correspond to the variance parameter of the normal distribution or the dispersion parameter of the negative Binomial distribution.
- If `object` is an instance of the `bundle` class, a matrix is returned with row number equal to `batch_size` and column number equal to the number of emulators in the bundle, giving positions (i.e., row numbers) of next design points from `x_cand` to be added to individual emulators.

Note

- The column order of the first argument of `aggregate` must be consistent with the order of emulator output dimensions (if `object` is an instance of the `dgp` class), or the order of emulators placed in `object` if `object` is an instance of the `bundle` class;
- If `x_cand` is supplied as a list when `object` is an instance of `bundle` class and a `aggregate` function is provided, the matrices in `x_cand` must have common rows (i.e., the candidate sets of emulators in the bundle have common input locations) so the `aggregate` function can be applied.
- Any R vector detected in `x_cand` will be treated as a column vector and automatically converted into a single-column R matrix.

References

Beck, J., & Guillas, S. (2016). Sequential design with mutual information for computer experiments (MICE): emulation of a tsunami model. *SIAM/ASA Journal on Uncertainty Quantification*, **4(1)**, 739-766.

Examples

```
## Not run:

# load packages and the Python env
library(lhs)
library(dgpsi)

# construct a 1D non-stationary function
f <- function(x) {
 sin(30*((2*x-1)/2-0.4)^5)*cos(20*((2*x-1)/2-0.4))
}
```

```
# generate the initial design
X <- maximinLHS(10,1)
Y <- f(X)

# training a 2-layered DGP emulator with the global connection off
m <- dgp(X, Y, connect = F)

# generate a candidate set
x_cand <- maximinLHS(200,1)

# locate the next design point using MICE
next_point <- mice(m, x_cand = x_cand)
X_new <- x_cand[next_point,,drop = F]

# obtain the corresponding output at the located design point
Y_new <- f(X_new)

# combine the new input-output pair to the existing data
X <- rbind(X, X_new)
Y <- rbind(Y, Y_new)

# update the DGP emulator with the new input and output data and refit
m <- update(m, X, Y, refit = TRUE)

# plot the LOO validation
plot(m)

## End(Not run)
```

NegBin *Initialize a negative Binomial likelihood node*

Description

This function constructs a likelihood object to represent a negative Binomial likelihood node.

Usage

```
NegBin(input_dim = NULL)
```

Arguments

input_dim	a vector of length two that contains the indices of two GP nodes in the feeding layer whose outputs feed into this likelihood node. When set to NULL, all outputs from GP nodes in the feeding layer feed into this likelihood node, and in such a case one needs to ensure that only two GP nodes are specified in the feeding layer. Defaults to NULL.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

A 'python' object to represent a negative Binomial likelihood node.

Note

The negative Binomial likelihood node can only be linked to two feeding GP nodes.

Examples

```
## Not run:

# Check https://mingdeyu.github.io/dgpsi-R/ for examples
# on how to customize DGP structures using NegBin().

## End(Not run)
```

`nllik` *Calculate negative predicted log-likelihood*

Description

This function computes the negative predicted log-likelihood from a DGP emulator with a likelihood layer.

Usage

```
nllik(object, x, y)
```

Arguments

`object`	an instance of the dgp class and it should be produced by `dgp()` with one of the following two settings: 1. if `struc` = `NULL`, `likelihood` is not `NULL`; 2. if a customized structure is provided to `struc`, the final layer must be likelihood layer containing only one likelihood node produced by `Poisson()`, `Hetero()`, or `NegBin()`.
`x`	a matrix where each row is an input testing data point and each column is an input dimension.
`y`	a matrix with only one column where each row is a scalar-valued testing output data point.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

An updated `object` is returned with an additional slot named `NLL` that contains two elements. The first one, named `meanNLL`, is a scalar that gives the average negative predicted log-likelihood across all testing data points. The second one, named `allNLL`, is a vector that gives the negative predicted log-likelihood for each testing data point.

Note

Any R vector detected in `x` and `y` will be treated as a column vector and automatically converted into a single-column R matrix. Thus, if `x` is a single testing data point with multiple dimensions, it must be given as a matrix.

Examples

```
## Not run:

# Check https://mingdeyu.github.io/dgpsi-R/ for examples
# on how to compute the negative predicted log-likelihood
# using nllik().

## End(Not run)
```

---

pack *Pack GP and DGP emulators into a bundle*

Description

This function packs GP emulators and DGP emulators into a `bundle` class for sequential designs if each emulator emulates one output dimension of the underlying simulator.

Usage

```
pack(..., id = NULL)
```

Arguments

`...`	a sequence or a list of emulators produced by `gp()` or `dgp()`.
`id`	an ID to be assigned to the bundle emulator. If an ID is not provided (i.e., `id = NULL`), a UUID (Universally Unique Identifier) will be automatically generated and assigned to the emulator. Default to `NULL`.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

An S3 class named bundle to be used by design() for sequential designs. It has:

- a slot called id that is assigned through the id argument.
- *N* slots named emulator1,...,emulatorN, each of which contains a GP or DGP emulator, where *N* is the number of emulators that are provided to the function.
- a slot called data which contains two elements X and Y. X contains *N* matrices named emulator1,...,emulatorN that are training input data for different emulators. Y contains *N* single-column matrices named emulator1,...,emulatorN that are training output data for different emulators.

Examples

```
## Not run:

# load packages and the Python env
library(lhs)
library(dgpsi)

# construct a function with a two-dimensional output
f <- function(x) {
 y1 = sin(30*((2*x-1)/2-0.4)^5)*cos(20*((2*x-1)/2-0.4))
 y2 = 1/3*sin(2*(2*x - 1))+2/3*exp(-30*(2*(2*x-1))^2)+1/3
 return(cbind(y1,y2))
}

# generate the initial design
X <- maximinLHS(10,1)
Y <- f(X)

# generate the validation data
validate_x <- maximinLHS(30,1)
validate_y <- f(validate_x)

# training a 2-layered DGP emulator with respect to each output with the global connection off
m1 <- dgp(X, Y[,1], connect=F)
m2 <- dgp(X, Y[,2], connect=F)

# specify the range of the input dimension
lim <- c(0, 1)

# pack emulators to form an emulator bundle
m <- pack(m1, m2)

# 1st wave of the sequential design with 10 steps with target RMSE 0.01
m <- design(m, N=10, limits = lim, f = f, x_test = validate_x, y_test = validate_y, target = 0.01)

# 2rd wave of the sequential design with 10 steps, the same target, and the aggregation
# function that takes the average of the criterion scores across the two outputs
g <- function(x){
  return(rowMeans(x))
}
```

```
m <- design(m, N=10, limits = lim, f = f, x_test = validate_x,
                    y_test = validate_y, aggregate = g, target = 0.01)

# draw sequential designs of the two packed emulators
draw(m, emulator = 1, type = 'design')
draw(m, emulator = 2, type = 'design')

# inspect the traces of RMSEs of the two packed emulators
draw(m, emulator = 1, type = 'rmse')
draw(m, emulator = 2, type = 'rmse')

# write and read the constructed emulator bundle
write(m, 'bundle_dgp')
m <- read('bundle_dgp')

# unpack the bundle into individual emulators
m_unpacked <- unpack(m)

# plot OOS validations of individual emulators
plot(m_unpacked[[1]], x_test = validate_x, y_test = validate_y[,1])
plot(m_unpacked[[2]], x_test = validate_x, y_test = validate_y[,2])

## End(Not run)
```

---

pei *Locate the next design point for a (D)GP emulator or a bundle of (D)GP emulators using PEI*

Description

This function searches from a candidate set to locate the next design point(s) to be added to a (D)GP emulator or a bundle of (D)GP emulators using the Pseudo Expected Improvement (PEI), see the reference below.

Usage

```
pei(object, x_cand, ...)

## S3 method for class 'gp'
pei(
  object,
  x_cand,
  pseudo_points = NULL,
  batch_size = 1,
  M = 50,
  workers = 1,
  ...
)
```

```
## S3 method for class 'dgp'
pei(
  object,
  x_cand,
  pseudo_points = NULL,
  batch_size = 1,
  M = 50,
  workers = 1,
  aggregate = NULL,
  ...
)

## S3 method for class 'bundle'
pei(
  object,
  x_cand,
  pseudo_points = NULL,
  batch_size = 1,
  M = 50,
  workers = 1,
  aggregate = NULL,
  ...
)
```

Arguments

`object`	can be one of the following: • the S3 class gp. • the S3 class dgp. • the S3 class bundle.
`x_cand`	a matrix (with each row being a design point and column being an input dimension) that gives a candidate set from which the next design point(s) are determined. If `object` is an instance of the `bundle` class, `x_cand` could also be a list with the length equal to the number of emulators contained in the `object`. Each slot in `x_cand` is a matrix that gives a candidate set for each emulator included in the bundle. See *Note* section below for further information.
`...`	any arguments (with names different from those of arguments used in `pei()`) that are used by `aggregate` or `gp()` (for emulating the ES-LOO errors) can be passed here.
`pseudo_points`	an optional matrix (with columns being input dimensions) that gives the pseudo input points for PEI calculations. See the reference below for further details about the pseudo points. When `object` is an instance of the `bundle` class, `pseudo_points` can also be a list with the length equal to the number of emulators in the bundle. Each element in the list is a matrix that gives the the pseudo input points for the corresponding emulator in the bundle. Defaults to `NULL`. When `pei()` is used in `design()`, `pseudo_points` will be automatically generated by `design()`.

`batch_size` an integer that gives the number of design points to be chosen. Defaults to `1`.

`M` the size of the conditioning set for the Vecchia approximation in the criterion calculation. This argument is only used if the emulator `object` was constructed under the Vecchia approximation. Defaults to `50`.

`workers` the number of processes to be used for the criterion calculation. If set to `NULL`, the number of processes is set to `max physical cores available %/% 2`. Defaults to `1`.

`aggregate` an R function that aggregates scores of the PEI across different output dimensions (if `object` is an instance of the `dgp` class) or across different emulators (if `object` is an instance of the `bundle` class). The function should be specified in the following basic form:

- the first argument is a matrix representing scores. The rows of the matrix correspond to different design points. The number of columns of the matrix equals to:
  - the emulator output dimension if `object` is an instance of the `dgp` class; or
  - the number of emulators contained in `object` if `object` is an instance of the `bundle` class.
- the output should be a vector that gives aggregations of scores at different design points.

Set to `NULL` to disable the aggregation. Defaults to `NULL`.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

- If `object` is an instance of the `gp` class, a vector is returned with the length equal to `batch_size`, giving the positions (i.e., row numbers) of next design points from `x_cand`.
- If `object` is an instance of the `dgp` class, a matrix is returned with row number equal to `batch_size` and column number equal to one (if `aggregate` is not `NULL`) or the output dimension (if `aggregate` is `NULL`), giving positions (i.e., row numbers) of next design points from `x_cand` to be added to the DGP emulator across different outputs.
- If `object` is an instance of the `bundle` class, a matrix is returned with row number equal to `batch_size` and column number equal to the number of emulators in the bundle, giving positions (i.e., row numbers) of next design points from `x_cand` to be added to individual emulators.

Note

- The column order of the first argument of `aggregate` must be consistent with the order of emulator output dimensions (if `object` is an instance of the `dgp` class), or the order of emulators placed in `object` if `object` is an instance of the `bundle` class;
- If `x_cand` is supplied as a list when `object` is an instance of `bundle` class and a `aggregate` function is provided, the matrices in `x_cand` must have common rows (i.e., the candidate sets

of emulators in the bundle have common input locations) so the `aggregate` function can be applied.

- The function is only applicable to DGP emulators without likelihood layers.
- Any R vector detected in `x_cand` and `pseudo_points` will be treated as a column vector and automatically converted into a single-column R matrix.

References

Mohammadi, H., Challenor, P., Williamson, D., & Goodfellow, M. (2022). Cross-validation-based adaptive sampling for Gaussian process models. *SIAM/ASA Journal on Uncertainty Quantification*, **10(1)**, 294-316.

Examples

```
## Not run:

# load packages and the Python env
library(lhs)
library(dgpsi)

# construct a 1D non-stationary function
f <- function(x) {
 sin(30*((2*x-1)/2-0.4)^5)*cos(20*((2*x-1)/2-0.4))
}

# generate the initial design
X <- maximinLHS(10,1)
Y <- f(X)

# training a 2-layered DGP emulator with the global connection off
m <- dgp(X, Y, connect = F)

# generate a candidate set
x_cand <- maximinLHS(200,1)

# locate the next design point using PEI
next_point <- pei(m, x_cand = x_cand)
X_new <- x_cand[next_point,,drop = F]

# obtain the corresponding output at the located design point
Y_new <- f(X_new)

# combine the new input-output pair to the existing data
X <- rbind(X, X_new)
Y <- rbind(Y, Y_new)

# update the DGP emulator with the new input and output data and refit
m <- update(m, X, Y, refit = TRUE)

# plot the LOO validation
plot(m)
```

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

---

plot *Validation plots of a constructed GP, DGP, or linked (D)GP emulator*

Description

This function draws validation plots of a GP, DGP, or linked (D)GP emulator.

Usage

```
## S3 method for class 'dgp'
plot(
  x,
  x_test = NULL,
  y_test = NULL,
  dim = NULL,
  method = "mean_var",
  style = 1,
  min_max = TRUE,
  color = "turbo",
  type = "points",
  verb = TRUE,
  M = 50,
  force = FALSE,
  cores = 1,
  ...
)

## S3 method for class 'lgp'
plot(
  x,
  x_test = NULL,
  y_test = NULL,
  dim = NULL,
  method = "mean_var",
  style = 1,
  min_max = TRUE,
  color = "turbo",
  type = "points",
  M = 50,
  verb = TRUE,
  force = FALSE,
  cores = 1,
  ...
)
```

```
## S3 method for class 'gp'
plot(
  x,
  x_test = NULL,
  y_test = NULL,
  dim = NULL,
  method = "mean_var",
  style = 1,
  min_max = TRUE,
  color = "turbo",
  type = "points",
  verb = TRUE,
  M = 50,
  force = FALSE,
  cores = 1,
  ...
)
```

Arguments

x	can be one of the following emulator classes: • the S3 class gp. • the S3 class dgp. • the S3 class lgp.
x_test	same as that of validate().
y_test	same as that of validate().
dim	if dim = NULL, the index of an emulator's input will be shown on the x-axis in validation plots. Otherwise, dim indicates which dimension of an emulator's input will be shown on the x-axis in validation plots: • If x is an instance of the gp of dgp class, dim is an integer. • If x is an instance of the lgp class, dim can be 1. an integer referring to the dimension of the global input to emulators in the first layer of a linked emulator system; or 2. a vector of three integers referring to the dimension (specified by the third integer) of the global input to an emulator (specified by the second integer) in a layer (specified by the first integer) that is not the first layer of a linked emulator system. This argument is only used when style = 1 and the emulator input is at least two-dimensional. Defaults to NULL.
method	same as that of validate().
style	either 1 or 2, indicating two different types of validation plots.
min_max	a bool indicating if min-max normalization will be used to scale the testing output, RMSE, predictive mean and std from the emulator. Defaults to TRUE.
color	a character string indicating the color map to use when style = 2: • 'magma' (or 'A')

- 'inferno' (or 'B')
- 'plasma' (or 'C')
- 'viridis' (or 'D')
- 'cividis' (or 'E')
- 'rocket' (or 'F')
- 'mako' (or 'G')
- 'turbo' (or 'H')

Defaults to 'turbo' (or 'H').

type	either 'line' or 'points', indicating whether to draw testing data in the OOS validation plot as a line or individual points when the input of the emulator is one-dimensional and style = 1. Defaults to 'points'
verb	a bool indicating if the trace information on plotting will be printed during the function execution. Defaults to TRUE.
M	same as that of validate().
force	same as that of validate().
cores	same as that of validate().
...	N/A.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

A patchwork object.

Note

- plot() calls validate() internally to obtain validation results for plotting. However, plot() will not export the emulator object with validation results. Instead, it only returns the plotting object. For small-scale validations (i.e., small training or testing data points), direct execution of plot() is fine. However, for moderate- to large-scale validations, it is recommended to first run validate() to obtain and store validation results in the emulator object, and then supply the object to plot(). This is because if an emulator object has the validation results stored, each time when plot() is invoked, unnecessary evaluations of repetitive LOO or OOS validation will not be implemented.
- plot() uses information provided in x_test and y_test to produce the OOS validation plots. Therefore, if validation results are already stored in x, unless x_test and y_test are identical to those used by validate(), plot() will re-evaluate OOS validations before plotting.
- Any R vector detected in x_test and y_test will be treated as a column vector and automatically converted into a single-column R matrix. Thus, if x_test or y_test is a single testing data point with multiple dimensions, it must be given as a matrix.
- The returned patchwork object contains the ggplot2 objects. One can modify the included individual ggplots by accessing them with double-bracket indexing. See https://patchwork.data-imaginist.com/ for further information.

Examples

```
## Not run:

# See gp(), dgp(), or lgp() for an example.

## End(Not run)
```

`Poisson` *Initialize a Poisson likelihood node*

Description

This function constructs a likelihood object to represent a Poisson likelihood node.

Usage

```
Poisson(input_dim = NULL)
```

Arguments

`input_dim`	a vector of length one that contains the indices of one GP node in the feeding layer whose outputs feed into this likelihood node. When set to `NULL`, all outputs from GP nodes in the feeding layer feed into this likelihood node, and in such a case one needs to ensure that only one GP node is specified in the feeding layer. Defaults to `NULL`.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

A 'python' object to represent a Poisson likelihood node.

Note

The Poisson likelihood node can only be linked to one feeding GP node.

Examples

```
## Not run:

# Check https://mingdeyu.github.io/dgpsi-R/ for examples
# on how to customize DGP structures using Poisson().

## End(Not run)
```

predict *Predictions from GP, DGP, or linked (D)GP emulators*

Description

This function implements single-core or multi-core predictions (with or without multi-threading) from GP, DGP, or linked (D)GP emulators.

Usage

```
## S3 method for class 'dgp'
predict(
  object,
  x,
  method = "mean_var",
  full_layer = FALSE,
  sample_size = 50,
  M = 50,
  cores = 1,
  chunks = NULL,
  ...
)

## S3 method for class 'lgp'
predict(
  object,
  x,
  method = "mean_var",
  full_layer = FALSE,
  sample_size = 50,
  M = 50,
  cores = 1,
  chunks = NULL,
  ...
)

## S3 method for class 'gp'
predict(
  object,
  x,
  method = "mean_var",
  sample_size = 50,
  M = 50,
  cores = 1,
  chunks = NULL,
  ...
)
```

Arguments

`object`	an instance of the `gp`, `dgp`, or `lgp` class.
`x`	the testing input data: • if `object` is an instance of the `gp` or `dgp` class, `x` is a matrix where each row is an input testing data point and each column is an input dimension. • if `object` is an instance of the `lgp` class, `x` can be a matrix or a list: – if `x` is a matrix, it is the global testing input data that feed into the emulators in the first layer of a system. The rows of `x` represent different input data points and the columns represent input dimensions across all emulators in the first layer of the system. In this case, it is assumed that the only global input to the system is the input to the emulators in the first layer and there is no global input to emulators in other layers. – if `x` is a list, it should have L (the number of layers in an emulator system) elements. The first element is a matrix that represents the global testing input data that feed into the emulators in the first layer of the system. The remaining *L-1* elements are *L-1* sub-lists, each of which contains a number (the same number of emulators in the corresponding layer) of matrices (rows being testing input data points and columns being input dimensions) that represent the global testing input data to the emulators in the corresponding layer. The matrices must be placed in the sub-lists based on how their corresponding emulators are placed in `struc` argument of `lgp()`. If there is no global input data to a certain emulator, set `NULL` in the corresponding sub-list of `x`.
`method`	the prediction approach: mean-variance (`"mean_var"`) or sampling (`"sampling"`) approach. Defaults to `"mean_var"`.
`full_layer`	a bool indicating whether to output the predictions of all layers. Defaults to `FALSE`. Only used when `object` is a DGP and linked (D)GP emulator.
`sample_size`	the number of samples to draw for each given imputation if `method = "sampling"`. Defaults to `50`.
`M`	the size of the conditioning set for the Vecchia approximation in the emulator prediction. Defaults to `50`. This argument is only used if the emulator `object` was constructed under the Vecchia approximation.
`cores`	the number of processes to be used for predictions. If set to `NULL`, the number of processes is set to `max physical cores available %/% 2`. Defaults to `1`.
`chunks`	the number of chunks that the testing input matrix `x` will be divided into for multi-cores to work on. Only used when `cores` is not `1`. If not specified (i.e., `chunks = NULL`), the number of chunks is set to the value of `cores`. Defaults to `NULL`.
`...`	N/A.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

- If `object` is an instance of the `gp` class:
  1. if `method = "mean_var"`: an updated `object` is returned with an additional slot called `results` that contains two matrices named `mean` for the predictive means and `var` for the predictive variances. Each matrix has only one column with its rows corresponding to testing positions (i.e., rows of `x`).
  2. if `method = "sampling"`: an updated `object` is returned with an additional slot called `results` that contains a matrix whose rows correspond to testing positions and columns correspond to `sample_size` number of samples drawn from the predictive distribution of GP.
- If `object` is an instance of the `dgp` class:
  1. if `method = "mean_var"` and `full_layer = FALSE`: an updated `object` is returned with an additional slot called `results` that contains two matrices named `mean` for the predictive means and `var` for the predictive variances respectively. Each matrix has its rows corresponding to testing positions and columns corresponding to DGP global output dimensions (i.e., the number of GP/likelihood nodes in the final layer).
  2. if `method = "mean_var"` and `full_layer = TRUE`: an updated `object` is returned with an additional slot called `results` that contains two sub-lists named `mean` for the predictive means and `var` for the predictive variances respectively. Each sub-list contains L (i.e., the number of layers) matrices named `layer1, layer2,..., layerL`. Each matrix has its rows corresponding to testing positions and columns corresponding to output dimensions (i.e., the number of GP/likelihood nodes from the associated layer).
  3. if `method = "sampling"` and `full_layer = FALSE`: an updated `object` is returned with an additional slot called `results` that contains D (i.e., the number of GP/likelihood nodes in the final layer) matrices named `output1, output2,..., outputD`. Each matrix has its rows corresponding to testing positions and columns corresponding to samples of size: `B * sample_size`, where `B` is the number of imputations specified in `dgp()`.
  4. if `method = "sampling"` and `full_layer = TRUE`: an updated `object` is returned with an additional slot called `results` that contains L (i.e., the number of layers) sub-lists named `layer1, layer2,..., layerL`. Each sub-list represents samples drawn from the GP/likelihood nodes in the corresponding layer, and contains D (i.e., the number of GP/likelihood nodes in the corresponding layer) matrices named `output1, output2,..., outputD`. Each matrix gives samples of the output from one of D GP/likelihood nodes, and has its rows corresponding to testing positions and columns corresponding to samples of size: `B * sample_size`, where `B` is the number of imputations specified in `dgp()`.
- If `object` is an instance of the `lgp` class:
  1. if `method = "mean_var"` and `full_layer = FALSE`: an updated `object` is returned with an additional slot called `results` that contains two sub-lists named `mean` for the predictive means and `var` for the predictive variances respectively. Each sub-list contains K number (same number of emulators in the final layer of the system) of matrices named `emulator1, emulator2,..., emulatorM`. Each matrix has its rows corresponding to global testing positions and columns corresponding to output dimensions of the associated emulator in the final layer.
  2. if `method = "mean_var"` and `full_layer = TRUE`: an updated `object` is returned with an additional slot called `results` that contains two sub-lists named `mean` for the predictive means and `var` for the predictive variances respectively. Each sub-list contains L (i.e., the

number of layers in the emulated system) components named `layer1, layer2,..., layerL`. Each component represents a layer and contains K number (same number of emulators in the corresponding layer of the system) of matrices named `emulator1, emulator2,..., emulatorM`. Each matrix has its rows corresponding to global testing positions and columns corresponding to output dimensions of the associated GP/DGP emulator in the corresponding layer.

3. if `method = "sampling"` and `full_layer = FALSE`: an updated `object` is returned with an additional slot called `results` that contains K number (same number of emulators in the final layer of the system) of sub-lists named `emulator1, emulator2,..., emulatorM`. Each sub-list corresponds to an emulator in the final layer, and contains D matrices, named `output1, output2,..., outputD`, that correspond to the output dimensions of the GP/DGP emulator. Each matrix has its rows corresponding to testing positions and columns corresponding to samples of size: `B * sample_size`, where `B` is the number of imputations specified in `lgp()`.
4. if `method = "sampling"` and `full_layer = TRUE`: an updated `object` is returned with an additional slot called `results` that contains L (i.e., the number of layers of the emulated system) sub-lists named `layer1, layer2,..., layerL`. Each sub-list represents a layer and contains K number (same number of emulators in the corresponding layer of the system) of components named `emulator1, emulator2,..., emulatorM`. Each component corresponds to an emulator in the associated layer, and contains D matrices, named `output1, output2,..., outputD`, that correspond to the output dimensions of the GP/DGP emulator. Each matrix has its rows corresponding to testing positions and columns corresponding to samples of size: `B * sample_size`, where `B` is the number of imputations specified in `lgp()`.

The `results` slot will also include the value of `M`, which represents the size of the conditioning set for the Vecchia approximation, if used, in the emulator prediction.

Note

Any R vector detected in `x` will be treated as a column vector and automatically converted into a single-column R matrix. Thus, if `x` is a single testing data point with multiple dimensions, it must be given as a matrix.

Examples

```
## Not run:

# See gp(), dgp(), or lgp() for an example.

## End(Not run)
```

`prune` *Static pruning of a DGP emulator*

Description

This function implements the static pruning of a DGP emulator.

Usage

```
prune(object, control = list(), verb = TRUE)
```

Arguments

object	an instance of the dgp class that is generated by dgp() with struc = NULL.
control	a list that can supply the following two components to control the static pruning of the DGP emulator: • min_size, the minimum number of design points required to trigger the pruning. Defaults to 10 times of the input dimensions. • threshold, the R2 value above which a GP node is considered redundant and removable. Defaults to 0.97.
verb	a bool indicating if the trace information will be printed during the function execution. Defaults to TRUE.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

An updated object that could be an instance of gp, dgp, or bundle (of GP emulators) class.

Note

- The function requires a DGP emulator that has been trained with a dataset comprising a minimum size equal to min_size in control. If the training dataset size is smaller than this, it is suggested to enrich the design of the DGP emulator and prune its structure dynamically using the design() function. Depending on the design of the DGP emulator, the static pruning may not be accurate. It is thus suggested to implement dynamic pruning as a part of the sequential design via design().
- The following slots:
  - loo and oos created by validate(); and
  - results created by predict();

  in object will be removed and not contained in the returned object.

Examples

```
## Not run:

# load the package and the Python env
library(dgpsi)

# construct the borehole function over a hypercube
f <- function(x){
  x[,1] <- (0.15 - 0.5) * x[,1] + 0.5
  x[,2] <- exp((log(50000) - log(100)) * x[,2] + log(100))
  x[,3] <- (115600 - 63070) *x[,3] + 63070
```

```
  x[,4] <- (1110 - 990) * x[,4] + 990
  x[,5] <- (116 - 63.1) * x[,5] + 63.1
  x[,6] <- (820 - 700) * x[,6] + 700
  x[,7] <- (1680 - 1120) * x[,7] + 1120
  x[,8] <- (12045 - 9855) * x[,8] + 9855
  y <- apply(x, 1, RobustGaSP::borehole)
}

# set a random seed
set_seed(999)

# generate training data
X <- maximinLHS(80, 8)
Y <- f(X)

# generate validation data
validate_x <- maximinLHS(500, 8)
validate_y <- f(validate_x)

# training a DGP emulator with anisotropic squared exponential kernels
m <- dgp(X, Y, share = F)

# OOS validation of the DGP emulator
plot(m, validate_x, validate_y)

# prune the emulator until no more GP nodes are removable
m <- prune(m)

# OOS validation of the resulting emulator
plot(m, validate_x, validate_y)

## End(Not run)
```

`read` *Load the stored emulator*

Description

This function loads the `.pkl` file that stores the emulator.

Usage

```
read(pkl_file)
```

Arguments

`pkl_file`	the path to and the name of the `.pkl` file where the emulator is stored.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

The S3 class of a GP emulator, a DGP emulator, a linked (D)GP emulator, or a bundle of (D)GP emulators.

Examples

```
## Not run:

# See gp(), dgp(), lgp(), or pack() for an example.

## End(Not run)
```

---

set_imp *Reset number of imputations for a DGP emulator*

Description

This function resets the number of imputations for predictions from a DGP emulator.

Usage

```
set_imp(object, B = 5)
```

Arguments

object	an instance of the S3 class dgp.
B	the number of imputations to produce predictions from object. Increase the value to account for more imputation uncertainties with slower predictions. Decrease the value for lower imputation uncertainties but faster predictions. Defaults to 5.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

An updated object with the information of B incorporated.

Note

- This function is useful when a DGP emulator has been trained and one wants to make faster predictions by decreasing the number of imputations without rebuilding the emulator.
- The following slots:
  - loo and oos created by validate(); and
  - results created by predict() in object will be removed and not contained in the returned object.

Examples

```
## Not run:

# See design() for an example.

## End(Not run)
```

`set_linked_idx` *Set linked indices*

Description

This function adds the linked information to a GP or DGP emulator if the information is not provided when the emulator is constructed by `gp()` or `dgp()`.

Usage

```
set_linked_idx(object, idx)
```

Arguments

`object`	an instance of the S3 class gp or dgp.
`idx`	same as the argument `linked_idx` of `gp()` and `dgp()`.

Details

See further examples and tutorials at `https://mingdeyu.github.io/dgpsi-R/`.

Value

An updated `object` with the information of `idx` incorporated.

Note

This function is useful when different models are emulated by different teams. Each team can create their (D)GP emulator even without knowing how different emulators are connected together. When this information is available and different emulators are collected, the connection information between emulators can then be assigned to individual emulators with this function.

Examples

```
## Not run:

# See lgp() for an example.

## End(Not run)
```

set_seed *Random seed generator*

Description

This function initializes a random number generator that sets the random seed in both R and Python to ensure reproducible results from the package.

Usage

```
set_seed(seed)
```

Arguments

seed	a single integer value.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

No return value.

Examples

```
## Not run:

# See dgp() for an example.

## End(Not run)
```

set_thread_num *Set the number of threads*

Description

This function sets the number of threads for parallel computations involved in the package.

Usage

```
set_thread_num(num)
```

Arguments

num	the number of threads. If it is greater than the maximum number of threads available, the number of threads will be set to the maximum value.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

No return value.

`set_vecchia` *Add or remove the Vecchia approximation*

Description

This function adds or removes the Vecchia approximation from a GP, DGP or linked (D)GP emulator constructed by `gp()`, `dgp()` or `lgp()`.

Usage

```
set_vecchia(object, vecchia = TRUE, M = 25, ord = NULL)
```

Arguments

`object`	an instance of the S3 class `gp`, `dgp`, or `lgp`.
`vecchia`	a boolean or a list of booleans to indicate the addition or removal of the Vecchia approximation: • if `object` is an instance of the `gp` or `dgp` class, `vecchia` is a boolean that indicates either addition (`vecchia = TRUE`) or removal (`vecchia = FALSE`) of the Vecchia approximation from `object`. • if `object` is an instance of the `lgp` class, `x` can be a boolean or a list of booleans: – if `vecchia` is a boolean, it indicates either addition (`vecchia = TRUE`) or removal (`vecchia = FALSE`) of the Vecchia approximation from all individual (D)GP emulators contained in `object`. – if `vecchia` is a list of booleans, it should have same shape as `struc` that was supplied to `lgp()`. Each boolean in the list indicates if the corresponding (D)GP emulator contained in `object` shall have the Vecchia approximation added or removed.
`M`	the size of the conditioning set for the Vecchia approximation in the (D)GP emulator training. Defaults to `25`.
`ord`	an R function that returns the ordering of the input to the (D)GP emulator for the Vecchia approximation. The function must satisfy the following basic rules: • the first argument represents the lengthscale-scaled input to the GP emulator or the lengthscale-scaled input to a GP node of the DGP emulator. • the output of the function is a vector of indices that gives the ordering of the input to the GP emulator or the input to the GP nodes of the DGP emulator. If `ord = NULL`, the default random ordering is used. Defaults to `NULL`.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

An updated `object` with the Vecchia approximation either added or removed.

Note

This function is useful for quickly switching between Vecchia and non-Vecchia approximations for an existing emulator without the need to reconstruct the emulator. If the emulator was built without the Vecchia approximation, the function can add it, and if the emulator was built with the Vecchia approximation, the function can remove it. If the current state already matches the requested state, the emulator remains unchanged.

`summary` *Summary of a constructed GP, DGP, or linked (D)GP emulator*

Description

This function summarizes key information of a GP, DGP or linked (D)GP emulator.

Usage

```
## S3 method for class 'gp'
summary(object, ...)

## S3 method for class 'dgp'
summary(object, ...)

## S3 method for class 'lgp'
summary(object, ...)
```

Arguments

`object`	can be one of the following: • the S3 class gp. • the S3 class dgp. • the S3 class lgp.
`...`	N/A.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

A table summarizing key information contained in `object`.

Examples

```
## Not run:

# See gp(), dgp(), or lgp() for an example.

## End(Not run)
```

`trace_plot` *Plot of DGP model parameter traces*

Description

This function plots the traces of model parameters of a chosen GP node in a DGP emulator.

Usage

```
trace_plot(object, layer = NULL, node = 1)
```

Arguments

`object`	an instance of the dgp class.
`layer`	the index of a layer. Defaults to `NULL` for the final layer.
`node`	the index of a GP node in the layer specified by `layer`. Defaults to `1` for the first GP node in the corresponding layer.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

A `ggplot` object.

Examples

```
## Not run:

# See dgp() for an example.

## End(Not run)
```

unpack *Unpack a bundle of (D)GP emulators*

Description

This function unpacks a bundle of (D)GP emulators safely so any further manipulations of unpacked individual emulators will not impact the ones in the bundle.

Usage

```
unpack(object)
```

Arguments

`object`	an instance of the class `bundle`.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

A named list that contains individual emulators (named `emulator1,...,emulatorS`) packed in `object`, where `S` is the number of emulators in `object`.

Examples

```
## Not run:

# See pack() for an example.

## End(Not run)
```

update *Update a GP or DGP emulator*

Description

This function updates the training input and output of a GP or DGP emulator with an option to refit the emulator.

Usage

```
update(object, X, Y, refit, reset, verb, ...)

## S3 method for class 'dgp'
update(
  object,
  X,
  Y,
  refit = FALSE,
  reset = FALSE,
  verb = TRUE,
  N = NULL,
  cores = 1,
  ess_burn = 10,
  B = NULL,
  ...
)

## S3 method for class 'gp'
update(object, X, Y, refit = FALSE, reset = FALSE, verb = TRUE, ...)
```

Arguments

object	can be one of the following: • the S3 class gp. • the S3 class dgp.
X	the new input data which is a matrix where each row is an input training data point and each column is an input dimension.
Y	the new output data: • If object is an instance of the gp class, Y is a matrix with only one column and each row being an output data point. • If object is an instance of the dgp class, Y is a matrix with its rows being output data points and columns being output dimensions. When likelihood (see below) is not NULL, Y must be a matrix with only one column.
refit	a bool indicating whether to re-fit the emulator object after the training input and output are updated. Defaults to FALSE.
reset	a bool indicating whether to reset hyperparameters of the emulator object to their initial values when the emulator was constructed, after the training input and output are updated. Defaults to FALSE.
verb	a bool indicating if the trace information will be printed during the function execution. Defaults to TRUE.
...	N/A.
N	number of training iterations used to re-fit the emulator object if it is an instance of the dgp class. If set to NULL, the number of iterations is set to 100 if the DGP emulator was constructed without the Vecchia approximation, and is set to 50 if Vecchia approximation was used. Defaults to NULL.

cores — the number of processes to be used to re-fit GP components (in the same layer) at each M-step during the re-fitting. If set to NULL, the number of processes is set to (max physical cores available - 1) if vecchia = FALSE and max physical cores available %/% 2 if vecchia = TRUE. Only use multiple processes when there is a large number of GP components in different layers and optimization of GP components is computationally expensive. Defaults to 1.

ess_burn — number of burnin steps for the ESS-within-Gibbs at each I-step in training the emulator object if it is an instance of the dgp class. Defaults to 10.

B — the number of imputations for predictions from the updated emulator object if it is an instance of the dgp class. This overrides the number of imputations set in object. Set to NULL to use the same number of imputations set in object. Defaults to NULL.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

An updated object.

Note

- The following slots:
  - loo and oos created by validate();
  - results created by predict(); and
  - design created by design()

  in object will be removed and not contained in the returned object.
- Any R vector detected in X and Y will be treated as a column vector and automatically converted into a single-column R matrix. Thus, if X is a single data point with multiple dimensions, it must be given as a matrix.

Examples

```
## Not run:

# See alm(), mice(), pei(), or vigf() for an example.

## End(Not run)
```

validate *Validate a constructed GP, DGP, or linked (D)GP emulator*

Description

This function validate a constructed GP, DGP, or linked (D)GP emulator via the Leave-One-Out (LOO) cross validation or Out-Of-Sample (OOS) validation.

Usage

```
validate(object, x_test, y_test, method, verb, M, force, cores, ...)

## S3 method for class 'gp'
validate(
  object,
  x_test = NULL,
  y_test = NULL,
  method = "mean_var",
  verb = TRUE,
  M = 50,
  force = FALSE,
  cores = 1,
  ...
)

## S3 method for class 'dgp'
validate(
  object,
  x_test = NULL,
  y_test = NULL,
  method = "mean_var",
  verb = TRUE,
  M = 50,
  force = FALSE,
  cores = 1,
  ...
)

## S3 method for class 'lgp'
validate(
  object,
  x_test = NULL,
  y_test = NULL,
  method = "mean_var",
  verb = TRUE,
  M = 50,
  force = FALSE,
```

```
  cores = 1,
  ...
)
```

Arguments

`object`	can be one of the following: • the S3 class gp. • the S3 class dgp. • the S3 class lgp.
`x_test`	the OOS testing input data: • if x is an instance of the gp or dgp class, `x_test` is a matrix where each row is an input testing data point and each column is an input dimension. • if x is an instance of the lgp class, `x_test` can be a matrix or a list: – if `x_test` is a matrix, it is the global testing input data that feed into the emulators in the first layer of a system. The rows of `x_test` represent different input data points and the columns represent input dimensions across all emulators in the first layer of the system. In this case, it is assumed that the only global input to the system is the input to the emulators in the first layer and there is no global input to emulators in other layers. – if `x_test` is a list, it should have *L* (the number of layers in an emulator system) elements. The first element is a matrix that represents the global testing input data that feed into the emulators in the first layer of the system. The remaining *L-1* elements are *L-1* sub-lists, each of which contains a number (the same number of emulators in the corresponding layer) of matrices (rows being testing input data points and columns being input dimensions) that represent the global testing input data to the emulators in the corresponding layer. The matrices must be placed in the sub-lists based on how their corresponding emulators are placed in `struc` argument of `lgp()`. If there is no global input data to a certain emulator, set `NULL` in the corresponding sub-list of `x_test`. `x_test` must be provided for the validation if x is an instance of the lgp. Defaults to `NULL`.
`y_test`	the OOS testing output data that correspond to `x_test`: • if x is an instance of the gp class, `y_test` is a matrix with only one column and each row being an testing output data point. • if x is an instance of the dgp class, `y_test` is a matrix with its rows being testing output data points and columns being output dimensions. • if x is an instance of the lgp class, `y_test` can be a single matrix or a list of matrices: – if `y_test` is a single matrix, then there is only one emulator in the final layer of the linked emulator system and `y_test` represents the emulator's output with rows being testing positions and columns being output dimensions.

- if y_test is a list, then y_test should have *M* number (the same number of emulators in the final layer of the system) of matrices. Each matrix has its rows corresponding to testing positions and columns corresponding to output dimensions of the associated emulator in the final layer.

y_test must be provided for the validation if x is an instance of the lgp. Defaults to NULL.

method — the prediction approach in validations: mean-variance ("mean_var") or sampling ("sampling") approach. Defaults to "mean_var".

verb — a bool indicating if the trace information on validations will be printed during the function execution. Defaults to TRUE.

M — the size of the conditioning set for the Vecchia approximation in the emulator validation. This argument is only used if the emulator object was constructed under the Vecchia approximation. Defaults to 50.

force — a bool indicating whether to force the LOO or OOS re-evaluation when loo or oos slot already exists in object. When force = FALSE, validate() will try to determine automatically if the LOO or OOS re-evaluation is needed. Set force to TRUE when LOO or OOS re-evaluation is required. Defaults to FALSE.

cores — the number of processes to be used for validations. If set to NULL, the number of processes is set to max physical cores available %/% 2. Defaults to 1.

... — N/A.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

- If object is an instance of the gp class, an updated object is returned with an additional slot called loo (for LOO cross validation) or oos (for OOS validation) that contains:
  - two slots called x_train (or x_test) and y_train (or y_test) that contain the validation data points for LOO (or OOS).
  - a column matrix called mean, if method = "mean_var", or median, if method = "sampling", that contains the predictive means or medians of the GP emulator at validation positions.
  - three column matrices called std, lower, and upper that contain the predictive standard deviations and credible intervals of the GP emulator at validation positions. If method = "mean_var", the upper and lower bounds of a credible interval are two standard deviations above and below the predictive mean. If method = "sampling", the upper and lower bounds of a credible interval are 2.5th and 97.5th percentiles.
  - a numeric value called rmse that contains the root mean/median squared error of the GP emulator.
  - a numeric value called nrmse that contains the (min-max) normalized root mean/median squared error of the GP emulator. The min-max normalization is based on the maximum and minimum values of the validation outputs contained in y_train (or y_test).
  - an integer called M that contains size of the conditioning set used for the Vecchia approximation, if used, in the emulator validation.

The rows of matrices (mean, median, std, lower, and upper) correspond to the validation positions.

- If object is an instance of the dgp class, an updated object is returned with an additional slot called loo (for LOO cross validation) or oos (for OOS validation) that contains:
  - two slots called x_train (or x_test) and y_train (or y_test) that contain the validation data points for LOO (or OOS).
  - a matrix called mean, if method = "mean_var", or median, if method = "sampling", that contains the predictive means or medians of the DGP emulator at validation positions.
  - three matrices called std, lower, and upper that contain the predictive standard deviations and credible intervals of the DGP emulator at validation positions. If method = "mean_var", the upper and lower bounds of a credible interval are two standard deviations above and below the predictive mean. If method = "sampling", the upper and lower bounds of a credible interval are 2.5th and 97.5th percentiles.
  - a vector called rmse that contains the root mean/median squared errors of the DGP emulator across different output dimensions.
  - a vector called nrmse that contains the (min-max) normalized root mean/median squared errors of the DGP emulator across different output dimensions. The min-max normalization is based on the maximum and minimum values of the validation outputs contained in y_train (or y_test).
  - an integer called M that contains size of the conditioning set used for the Vecchia approximation, if used, in the emulator validation.

  The rows and columns of matrices (mean, median, std, lower, and upper) correspond to the validation positions and DGP emulator output dimensions, respectively.
- If object is an instance of the lgp class, an updated object is returned with an additional slot called oos (for OOS validation) that contains:
  - two slots called x_test and y_test that contain the validation data points for OOS.
  - a list called mean, if method = "mean_var", or median, if method = "sampling", that contains the predictive means or medians of the linked (D)GP emulator at validation positions.
  - three lists called std, lower, and upper that contain the predictive standard deviations and credible intervals of the linked (D)GP emulator at validation positions. If method = "mean_var", the upper and lower bounds of a credible interval are two standard deviations above and below the predictive mean. If method = "sampling", the upper and lower bounds of a credible interval are 2.5th and 97.5th percentiles.
  - a list called rmse that contains the root mean/median squared errors of the linked (D)GP emulator.
  - a list called nrmse that contains the (min-max) normalized root mean/median squared errors of the linked (D)GP emulator. The min-max normalization is based on the maximum and minimum values of the validation outputs contained in y_test.
  - an integer called M that contains size of the conditioning set used for the Vecchia approximation, if used, in the emulator validation.

  Each element in mean, median, std, lower, upper, rmse, and nrmse corresponds to a (D)GP emulator in the final layer of the linked (D)GP emulator.

Note

- When both `x_test` and `y_test` are `NULL`, the LOO cross validation will be implemented. Otherwise, OOS validation will be implemented. The LOO validation is only applicable to a GP or DGP emulator (i.e., `x` is an instance of the `gp` or `dgp` class). If a linked (D)GP emulator (i.e., `x` is an instance of the `lgp` class) is provided, `x_test` and `y_test` must also be provided for OOS validation.
- Any R vector detected in `x_test` and `y_test` will be treated as a column vector and automatically converted into a single-column R matrix. Thus, if `x_test` or `y_test` is a single testing data point with multiple dimensions, it must be given as a matrix.

Examples

```
## Not run:

# See gp(), dgp(), or lgp() for an example.

## End(Not run)
```

`vigf` *Locate the next design point for a (D)GP emulator or a bundle of (D)GP emulators using VIGF*

Description

This function searches from a candidate set to locate the next design point(s) to be added to a (D)GP emulator or a bundle of (D)GP emulators using the Variance of Improvement for Global Fit (VIGF). For VIGF on GP emulators, see the reference below.

Usage

```
vigf(object, x_cand, ...)

## S3 method for class 'gp'
vigf(object, x_cand, batch_size = 1, M = 50, workers = 1, ...)

## S3 method for class 'dgp'
vigf(
  object,
  x_cand,
  batch_size = 1,
  M = 50,
  workers = 1,
  aggregate = NULL,
  ...
)

## S3 method for class 'bundle'
```

```
vigf(
  object,
  x_cand,
  batch_size = 1,
  M = 50,
  workers = 1,
  aggregate = NULL,
  ...
)
```

Arguments

object	can be one of the following: • the S3 class gp. • the S3 class dgp. • the S3 class bundle.
x_cand	a matrix (with each row being a design point and column being an input dimension) that gives a candidate set from which the next design point(s) are determined. If object is an instance of the bundle class, x_cand could also be a list with the length equal to the number of emulators contained in the object. Each slot in x_cand is a matrix that gives a candidate set for each emulator included in the bundle. See *Note* section below for further information.
...	any arguments (with names different from those of arguments used in vigf()) that are used by aggregate can be passed here.
batch_size	an integer that gives the number of design points to be chosen. Defaults to 1.
M	the size of the conditioning set for the Vecchia approximation in the criterion calculation. This argument is only used if the emulator object was constructed under the Vecchia approximation. Defaults to 50.
workers	the number of processes to be used for the criterion calculation. If set to NULL, the number of processes is set to max physical cores available %/% 2. Defaults to 1.
aggregate	an R function that aggregates scores of the VIGF across different output dimensions (if object is an instance of the dgp class) or across different emulators (if object is an instance of the bundle class). The function should be specified in the following basic form: • the first argument is a matrix representing scores. The rows of the matrix correspond to different design points. The number of columns of the matrix equals to: – the emulator output dimension if object is an instance of the dgp class; or – the number of emulators contained in object if object is an instance of the bundle class. • the output should be a vector that gives aggregations of scores at different design points. Set to NULL to disable the aggregation. Defaults to NULL.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

- If `object` is an instance of the `gp` class, a vector is returned with the length equal to `batch_size`, giving the positions (i.e., row numbers) of next design points from `x_cand`.
- If `object` is an instance of the `dgp` class, a matrix is returned with row number equal to `batch_size` and column number equal to one (if `aggregate` is not `NULL`) or the output dimension (if `aggregate` is `NULL`), giving positions (i.e., row numbers) of next design points from `x_cand` to be added to the DGP emulator across different outputs. If `object` is a DGP emulator with either `Hetero` or `NegBin` likelihood layer, the returned matrix has two columns with the first column giving positions of next design points from `x_cand` that correspond to the mean parameter of the normal or negative Binomial distribution, and the second column giving positions of next design points from `x_cand` that correspond to the variance parameter of the normal distribution or the dispersion parameter of the negative Binomial distribution.
- If `object` is an instance of the `bundle` class, a matrix is returned with row number equal to `batch_size` and column number equal to the number of emulators in the bundle, giving positions (i.e., row numbers) of next design points from `x_cand` to be added to individual emulators.

Note

- The column order of the first argument of `aggregate` must be consistent with the order of emulator output dimensions (if `object` is an instance of the `dgp` class), or the order of emulators placed in `object` if `object` is an instance of the `bundle` class;
- If `x_cand` is supplied as a list when `object` is an instance of `bundle` class and a `aggregate` function is provided, the matrices in `x_cand` must have common rows (i.e., the candidate sets of emulators in the bundle have common input locations) so the `aggregate` function can be applied.
- Any R vector detected in `x_cand` will be treated as a column vector and automatically converted into a single-column R matrix.

References

Mohammadi, H., & Challenor, P. (2022). Sequential adaptive design for emulating costly computer codes. *arXiv:2206.12113*.

Examples

```
## Not run:

# load packages and the Python env
library(lhs)
library(dgpsi)

# construct a 1D non-stationary function
f <- function(x) {
```

```
 sin(30*((2*x-1)/2-0.4)^5)*cos(20*((2*x-1)/2-0.4))
}

# generate the initial design
X <- maximinLHS(10,1)
Y <- f(X)

# training a 2-layered DGP emulator with the global connection off
m <- dgp(X, Y, connect = F)

# generate a candidate set
x_cand <- maximinLHS(200,1)

# locate the next design point using VIGF
next_point <- vigf(m, x_cand = x_cand)
X_new <- x_cand[next_point,,drop = F]

# obtain the corresponding output at the located design point
Y_new <- f(X_new)

# combine the new input-output pair to the existing data
X <- rbind(X, X_new)
Y <- rbind(Y, Y_new)

# update the DGP emulator with the new input and output data and refit
m <- update(m, X, Y, refit = TRUE)

# plot the LOO validation
plot(m)

## End(Not run)
```

`window` *Trim the sequences of model parameters of a DGP emulator*

Description

This function trim the sequences of model parameters of a DGP emulator that are generated during the training.

Usage

```
window(object, start, end = NULL, thin = 1)
```

Arguments

`object`	an instance of the S3 class `dgp`.
`start`	the first iteration before which all iterations are trimmed from the sequences.

end	the last iteration after which all iterations are trimmed from the sequences. Set to NULL to keep all iterations after (including) start. Defaults to NULL.
thin	the interval between the start and end iterations to thin out the sequences. Defaults to 1.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

An updated object with trimmed sequences of model parameters.

Note

- This function is useful when a DGP emulator has been trained and one wants to trim the sequences of model parameters and use the trimmed sequences to generate the point estimates of DGP model parameters for predictions.
- The following slots:
  - loo and oos created by validate(); and
  - results created by predict() in object will be removed and not contained in the returned object.

Examples

```
## Not run:

# See dgp() for an example.

## End(Not run)
```

---

write *Save the constructed emulator*

Description

This function saves the constructed emulator to a .pkl file.

Usage

```
write(object, pkl_file, light = TRUE)
```

Arguments

object	an instance of the S3 class gp, dgp, lgp, or bundle.
pkl_file	the path to and the name of the .pkl file to which the emulator object is saved.
light	a bool indicating if a light version of the constructed emulator (that requires a small storage) will be saved. This argument has no effects on GP or bundles of GP emulators. Defaults to TRUE.

Details

See further examples and tutorials at https://mingdeyu.github.io/dgpsi-R/.

Value

No return value. `object` will be save to a local `.pkl` file specified by `pkl_file`.

Note

Since the constructed emulators are 'python' objects, `save()` from R will not work as it is only for R objects. If `object` was processed by `set_vecchia()` to add or remove the Vecchia approximation, `light` needs to be set to `FALSE` to ensure reproducibility after the saved emulator is loaded by `read()`, since when `light = TRUE`, the imputations generated during emulator loading will be different.

Examples

```
## Not run:

# See gp(), dgp(), lgp(), or pack() for an example.

## End(Not run)
```

Index