Package 'dgpsi'

October 15, 2025

Type Package

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

Version 2.6.0

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) <doi:10.48550/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, clhs, dplyr, uuid, tidyr, rlang, lifecycle, magrittr, visNetwork, parallel, kableExtra

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

VignetteBuilder knitr

RoxygenNote 7.3.3

Language en-US

NeedsCompilation no

Author Deyu Ming [aut, cre, cph], Daniel Williamson [aut]

Repository CRAN

Date/Publication 2025-10-15 21:20:02 UTC

Contents

`alm` *Locate the next design point(s) for a (D)GP emulator or a bundle of (D)GP emulators using Active Learning MacKay (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) criterion (see the reference below).

Usage

```
alm(object, ...)

## S3 method for class 'gp'
alm(
  object,
  x_cand = NULL,
  n_start = 20,
  batch_size = 1,
  M = 50,
  workers = 1,
  limits = NULL,
  int = FALSE,
  ...
)

## S3 method for class 'dgp'
alm(
  object,
  x_cand = NULL,
  n_start = 20,
  batch_size = 1,
  M = 50,
  workers = 1,
  limits = NULL,
  int = FALSE,
  aggregate = NULL,
  ...
)

## S3 method for class 'bundle'
alm(
  object,
  x_cand = NULL,
  n_start = 20,
  batch_size = 1,
  M = 50,
  workers = 1,
  limits = NULL,
  int = FALSE,
  aggregate = NULL,
  ...
)
```

Arguments

object	can be one of the following: • the S3 class gp.

- the S3 class dgp.
- the S3 class bundle.

`...` any arguments (with names different from those of arguments used in `alm()`) that are used by `aggregate` can be passed here.

`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 and `aggregate` is not supplied, `x_cand` can also be a list. The list must have a length equal to the number of emulators in `object`, with each element being a matrix representing the candidate set for a corresponding emulator in the bundle. Defaults to `NULL`.

`n_start` an integer that gives the number of initial design points to be used to determine next design point(s). This argument is only used when `x_cand` is `NULL`. Defaults to `20`.

`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 design point selection. If set to `NULL`, the number of processes is set to `max physical cores available %/% 2`. Defaults to `1`. The argument does not currently support Windows machines when the `aggregate` function is provided, due to the significant overhead caused by initializing the Python environment for each worker under spawning.

`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. This argument is only used when `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 single 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. This argument is only used when `x_cand = NULL`. Defaults to `FALSE`.

`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 is equal 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 aggregate scores at different design points.

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

Details

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

Value

1. If `x_cand` is not `NULL`:
   - When `object` is an instance of the `gp` class, a vector of length `batch_size` is returned, containing the positions (row numbers) of the next design points from `x_cand`.
   - When `object` is an instance of the `dgp` class, a vector of length `batch_size * D` is returned, containing the positions (row numbers) of the next design points from `x_cand` to be added to the DGP emulator.
     - `D` is the number of output dimensions of the DGP emulator if no likelihood layer is included.
     - For a DGP emulator with a `Hetero` or `NegBin` likelihood layer, `D = 2`.
     - For a DGP emulator with a `Categorical` likelihood layer, `D = 1` for binary output or `D = K` for multi-class output with `K` classes.
   - When `object` is an instance of the `bundle` class, a matrix is returned with `batch_size` rows and a column for each emulator in the bundle, containing the positions (row numbers) of the next design points from `x_cand` for individual emulators.
2. If `x_cand` is `NULL`:
   - When `object` is an instance of the `gp` class, a matrix with `batch_size` rows is returned, giving the next design points to be evaluated.
   - When `object` is an instance of the `dgp` class, a matrix with `batch_size * D` rows is returned, where:
     - `D` is the number of output dimensions of the DGP emulator if no likelihood layer is included.
     - For a DGP emulator with a `Hetero` or `NegBin` likelihood layer, `D = 2`.
     - For a DGP emulator with a `Categorical` likelihood layer, `D = 1` for binary output or `D = K` for multi-class output with `K` classes.
   - When `object` is an instance of the `bundle` class, a list is returned with a length equal to the number of emulators in the bundle. Each element of the list is a matrix with `batch_size` rows, where each row represents a design point to be added to the corresponding emulator.

Note

The first column of the matrix supplied to the first argument of `aggregate` must correspond to the first output dimension of the DGP emulator if `object` is an instance of the `dgp` class, and so on for subsequent columns and dimensions. If `object` is an instance of the `bundle` class, the first column must correspond to the first emulator in the bundle, and so on for subsequent columns and emulators.

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)

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

# locate the next design point using ALM
X_new <- alm(m, limits = lim)

# 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)
```

`continue` *Continue training 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 to train the DGP emulator. 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 ESS-within-Gibbs at each I-step of the training. Defaults to 10.
verb	a bool indicating if a progress bar will be printed during training. 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 predictions. Increase the value to account for more imputation uncertainty. 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)
```

`deserialize` *Restore the serialized emulator*

Description

This function restores the serialized emulator created by `serialize()`.

Usage

```
deserialize(object)
```

Arguments

`object`	the serialized object of an emulator.

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.

Note

See the *Note* section in `serialize()`.

Examples

```
## Not run:

# See serialize() 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 sequential design and active learning for a (D)GP emulator or a bundle of (D)GP emulators, supporting an array of popular methods as well as user-specified approaches. It can also be used as a wrapper for Bayesian optimization methods.

Usage

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

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

```
  N,
  x_cand = NULL,
  y_cand = NULL,
  n_sample = 200,
  limits = NULL,
  f = NULL,
  reps = 1,
  freq = c(1, 1),
  x_test = NULL,
  y_test = NULL,
  reset = FALSE,
  target = NULL,
  method = vigf,
  batch_size = 1,
  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_sample = 200,
  limits = NULL,
  f = NULL,
  reps = 1,
  freq = c(1, 1),
  x_test = NULL,
  y_test = NULL,
  reset = FALSE,
  target = NULL,
  method = vigf,
  batch_size = 1,
  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_sample = 200,
  limits = NULL,
  f = NULL,
  reps = 1,
  freq = c(1, 1),
  x_test = NULL,
  y_test = NULL,
  reset = FALSE,
  target = NULL,
  method = vigf,
  batch_size = 1,
  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 iterations 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 from which the next design points are determined. 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_sample` an integer that gives 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`. If you provide a custom `method` function with an argument called `limits`, the value of `limits` will be passed to your function.

`f` an R function representing the simulator. `f` must adhere to the following rules:

- **First argument**: a matrix where rows correspond to different design points, and columns represent input dimensions.
- **Function output**:
  - a matrix where rows correspond to different outputs (matching the input design points) and columns represent output dimensions. If there is only one output dimension, the function should return a matrix with a single column.
  - alternatively, a list where:
    * the first element is the output matrix as described above.
    * additional named elements can optionally update values of arguments with matching names passed via `...`. This list output is useful if additional arguments to `f`, `method`, or `eval` need to be updated after each sequential design iteration.

See the *Note* section below for additional details. This argument is required and must be supplied 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` only if `f` is a stochastic function that can generate different responses given for the 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 indicating the number of iterations taken between re-estimating the emulator hyperparameters, and the second element defining the number of iterations to take between re-calculation of evaluating metrics on the validation set (see `x_test` below) via the `eval` function. 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 `freq[2]` iterations of the sequential design. Set to `NULL` for LOO-based emulator validation. Defaults to `NULL`. This argument is only used if `eval = NULL`.

`y_test` the testing output data corresponding to `x_test` for emulator validation after each `freq[2]` iterations 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 contains a testing output data point from the corresponding row of `x_test`.
- if `object` is an instance of the `dgp` class, `y_test` is a matrix with its rows containing testing output data points corresponding to the same rows of `x_test` and columns representing the output dimensions.
- if `object` is an instance of the `bundle` class, `y_test` is a matrix with each row representing the outputs for the corresponding row of `x_test` and each column representing the output of the different emulators in the bundle.

Set to `NULL` for 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 the hyperparameters of the emulator(s) to their initial values (as set during initial construction) before re-fitting. The re-fitting occurs based on the frequency specified by `freq[1]`. This option is useful when hyperparameters are suspected to have converged to a local optimum affecting validation performance.

- If a single bool is provided, it applies to every iteration of the sequential design.
- If a vector is provided, its length must equal `N` (even if the re-fit frequency specified in `freq[1]` is not 1) and it will apply to the corresponding iterations of the sequential design.

Defaults to `FALSE`.

`target` a number or vector specifying the target evaluation metric value(s) at which the sequential design should terminate. Defaults to `NULL`, in which case the sequential design stops after `N` steps. See the *Note* section below for further details about `target`.

`method` an R function that determines the next design points to be evaluated by `f`. The function must adhere to the following rules:

- **First argument**: an emulator object, which can be one of the following:
  - an instance of the `gp` class (produced by `gp()`);
  - an instance of the `dgp` class (produced by `dgp()`);
  - an instance of the `bundle` class (produced by `pack()`).
- **Second argument** (if `x_cand` is not `NULL`): a *candidate matrix* representing a set of potential design points from which the `method` function selects the next points.
- **Function output**:
  - If `x_cand` is not `NULL`:
    - for `gp` or `dgp` objects, the output must be a vector of row indices corresponding to the selected design points from the *candidate matrix* (the second argument).
    - for `bundle` objects, the output must be a matrix containing the row indices of the selected design points from the *candidate matrix*. Each column corresponds to the indices for an individual emulator in the bundle.

  - If x_cand is NULL:
    * for gp or dgp objects, the output must be a matrix where each row represents a new design point to be added.
    * for bundle objects, the output must be a list with a length equal to the number of emulators in the bundle. Each element in the list is a matrix where rows represent the new design points for the corresponding emulator.

See alm(), mice(), and vigf() for examples of built-in method functions. Defaults to vigf().

batch_size — an integer specifying the number of design points to select in a single iteration. Defaults to 1. This argument is used by the built-in method functions alm(), mice(), and vigf(). If you provide a custom method function with an argument named batch_size, the value of batch_size will be passed to your function.

eval — an R function that computes a customized metric for evaluating emulator performance. The function must adhere to the following rules:

- **First argument**: an emulator object, which can be one of the following:
  - an instance of the gp class (produced by gp());
  - an instance of the dgp class (produced by dgp());
  - an instance of the bundle class (produced by pack()).
- **Function output**:
  - for gp objects, the output must be a single metric value.
  - for dgp objects, the output can be a single metric value or a vector of metric values with a length equal to the number of output dimensions.
  - for bundle objects, the output can be a single metric value or a vector of metric values with a length equal to the number of emulators in the bundle.

If no custom function is provided, a built-in evaluation metric (RMSE or log-loss, in the case of DGP emulators with categorical likelihoods) will be used. Defaults to NULL. See the *Note* section below for additional details.

verb — a bool indicating if 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 automatic saving of the emulator during 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'.
- save_freq: an integer indicating the frequency of automatic saves, 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 save overwrites the previous one, keeping only the latest version. If `FALSE`, each automatic save creates a new file, preserving all previous versions. Defaults to `FALSE`.

`new_wave` a bool indicating whether the current call to `design()` will create a new wave of sequential designs or add the next sequence of designs to the most recent wave. This argument is relevant only if waves already exist in the emulator. Creating new waves can improve the visualization of sequential design performance across different calls to `design()` via `draw()`, and allows for specifying a different evaluation frequency in `freq`. However, disabling this option can help limit the number of waves visualized in `draw()` to avoid issues such as running out of distinct colors for large numbers of waves. 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 validation. 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 that differ from those used in `design()` but are required by `f`, `method`, or `eval` can be passed here. `design()` will forward relevant arguments to `f`, `method`, and `eval` based on the names of the additional arguments provided.

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

- If `train_N` is an integer, the DGP emulator will be re-fitted at each step (based on the re-fit frequency specified in `freq[1]`) using `train_N` iterations.
- If `train_N` is a vector, its length must be `N`, even if the re-fit frequency specified in `freq[1]` is not 1.
- If `train_N` is `NULL`, the DGP emulator will be re-fitted at each step (based on the re-fit frequency specified in `freq[1]`) using:
  - `100` iterations if the DGP emulator was constructed without the Vecchia approximation, or
  - `50` iterations if the 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()`. 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 dynamic pruning. Defaults to 10 times the number of input dimensions.
- `threshold`, the R^2 value above which a GP node is considered redundant. Defaults to `0.97`.
- `nexceed`, the minimum number of consecutive iterations that the R^2 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 design that have been applied to the emulator. Each slot contains the following elements:
  - `N`, an integer that gives the numbers of iterations implemented in the corresponding wave;
  - `rmse`, a matrix providing the evaluation metric values for emulators constructed during the corresponding wave, when `eval = NULL`. Each row of the matrix represents an iteration.
    - for an `object` of class `gp`, the matrix contains a single column of RMSE values.
    - for an `object` of class `dgp` without a categorical likelihood, each row contains mean/median squared errors corresponding to different output dimensions.
    - for an `object` of class `dgp` with a categorical likelihood, the matrix contains a single column of log-loss values.
    - for an `object` of class `bundle`, each row contains either mean/median squared errors or log-loss values for the emulators in the bundle.
  - `metric`: a matrix providing the values of custom evaluation metrics, as computed by the user-supplied `eval` function, for emulators constructed during the corresponding wave.
  - `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 evaluating metric computed by the `eval` or built-in function to stop the sequential design.
  - `reached`: indicates whether the `target` was reached at the end of the sequential design:
    - a bool if `object` is an instance of the `gp` or `dgp` class.

    * a vector of bools if `object` is an instance of the `bundle` class, with its length determined as follows:
      · equal to the number of emulators in the bundle when `eval = NULL`.
      · equal to the length of the output from `eval` when a custom `eval` function is provided.

- a slot called `type` that gives the type of validation:
  – 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 `x_cand` is supplied, 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 in later runs of `design()`.

See *Note* section below for further information.

Note

- Validation of an emulator is forced after the final step of a sequential design even if `N` is not a multiple 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 design, 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.

  For the 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 thresholds of evaluating metrics for different output dimensions or different emulators. If a single value is provided, it will be used as the 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` and `target` is given as a vector, 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 `NA`s. 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 design. Users should ensure that any errors within `f` are handled by appropriately returning `NA`s.
- 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.

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

[Updated]

This function builds and trains a DGP emulator.

Usage

```
dgp(
  X,
  Y,
  depth = 2,
  node = ncol(X),
  name = "sexp",
  lengthscale = 1,
  bounds = NULL,
  prior = "ga",
  share = TRUE,
  nugget_est = FALSE,
  nugget = NULL,
  scale_est = TRUE,
  scale = 1,
```

```
  connect = NULL,
  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,
  id = NULL,
  decouple = FALSE,
  link = "logit"
)
```

Arguments

X	a matrix where each row is an input training data point and each column represents an input dimension.
Y	a matrix containing observed training output data. The matrix has its rows being output data points and columns representing output dimensions. When likelihood (see below) is not NULL, Y must be a matrix with a single column.
depth	number of layers (including the likelihood layer) for a DGP structure. depth must be at least 2. Defaults to 2.
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).
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".
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`.

`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.

`prior` prior to be used for MAP estimation of 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"`.

`share` a bool indicating if all input dimensions of a GP node share a common lengthscale. Defaults to `TRUE`.

`nugget_est` **[Updated]** a bool or a bool vector indicating whether the nuggets of GP nodes in the final layer (or the layer feeding the likelihood node) should be estimated. If a bool is provided, it is applied to all GP nodes in that layer. If a bool vector is provided, its length must match the number of GP nodes:

- `ncol(Y)` if `likelihood = NULL`
- `2` if `likelihood` is `"Hetero"` or `"NegBin"`
- `1` if `likelihood` is `"Poisson"` or `"Categorical"` with two classes
- the number of classes if `likelihood` is `"Categorical"` with more than two classes.

Each element of the vector is applied to the corresponding GP node in the final layer (or the layer feeding the likelihood node). The value of a bool has following effects:

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

Defaults to `FALSE`.

`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 FALSE for deterministic emulation, where the emulator interpolates the training data points. Set nugget to a larger value and the bools in nugget_est to TRUE for stochastic emulation where the computer model outputs are assumed to follow a homogeneous Gaussian distribution. Defaults to 1e-6 if likelihood is NULL. If likelihood is not NULL, the nuggets of GPs that feed into the likelihood layer default to 1e-4, while those of all other GPs default to 1e-6.

scale_est — a bool or a bool vector indicating whether the variances of GP nodes in the final layer (or the layer feeding the likelihood node) should be estimated. If a bool is provided, it is applied to all GP nodes in that layer. If a bool vector is provided, its length must match the number of GP nodes:

- ncol(Y) if likelihood = NULL
- 2 if likelihood is "Hetero" or "NegBin"
- 1 if likelihood is "Poisson" or "Categorical" with two classes
- the number of classes if likelihood is "Categorical" with more than two classes.

The value of a bool has following effects:

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

Defaults to TRUE.

scale — the initial variance value(s) of GP nodes in the final layer (or the layer feeding the likelihood node). If it is a single numeric value, it will be applied to all GP nodes in the final layer (or the layer feeding the likelihood node). If it is a vector, its length must match the number of GP nodes:

- ncol(Y) if likelihood = NULL
- 2 if likelihood is "Hetero" or "NegBin"
- 1 if likelihood is "Poisson" or "Categorical" with two classes
- the number of classes if likelihood is "Categorical" with more than two classes.

Each numeric in the vector will be applied to the corresponding GP node.

Defaults to 1.

connect — a bool indicating whether to apply global input connections in the DGP structure. Setting this to FALSE may yield a better emulator in some cases. When set to NULL, the value defaults to FALSE if likelihood = "Categorical" and to TRUE otherwise. Defaults to 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 input-dependent noise).

3. "Poisson": a Poisson likelihood layer is added for emulation where the computer model outputs are counts and a Poisson distribution is used to model them.
4. "NegBin": a negative Binomial likelihood layer is added for emulation where the computer model outputs are counts and a negative Binomial distribution is used to capture dispersion variability in input space.
5. "Categorical": a categorical likelihood layer is added for emulation (classification), where the computer model output is categorical.

Defaults to 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 or apply predict() to check its emulation performance before 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 for 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 Gibbs steps, 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 used to produce predictions. Increase the value to refine the representation of imputation uncertainty. Defaults to 10.
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.
decouple	**[New]** A boolean indicating whether the model parameters for the heteroskedastic Gaussian likelihood, negative Binomial likelihood, and categorical likelihood (when the number of categories is greater than 2) should be modeled using separate deep Gaussian process hierarchies when depth is greater than 2. Defaults to FALSE.
link	**[New]** the link function used for binary classification when likelihood = "Categorical". Supported options are "logit" and "probit". Defaults to "logit".

Details

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

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, NegBin, or Categorical) 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. seed: the random seed generated to produce imputations. This information is stored for reproducibility when the DGP emulator (that was saved by write() with the light option light = TRUE) is loaded back to R by read().
  3. B: the number of imputations used to generate the emulator.
  4. vecchia: whether the Vecchia approximation is used for the GP emulator training.

5. `M`: the size of the conditioning set for the Vecchia approximation in the DGP emulator training. `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.
- `design()` for sequential design.
- `update()` to update the DGP emulator with new inputs and outputs.
- `alm()`, `mice()`, 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 and diagnostic plots for a sequential design*

Description

This function draws diagnostic and validation plots for a 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, type = "rmse", log = FALSE, emulator = NULL, ...)
```

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	specifies the type of plot or visualization to generate: • "rmse": generates a trace plot of RMSEs, log-losses for DGP emulators with categorical likelihoods, or custom evaluation metrics specified via the "eval" argument in the [design()] function. • "design": shows visualizations of input designs created by the sequential design procedure. Defaults to "rmse".
log	a bool indicating whether to plot RMSEs, log-losses (for DGP emulators with categorical likelihoods), or custom evaluation metrics on a log scale when type = "rmse". Defaults to FALSE.
emulator	an index or vector of indices of emulators packed in object. This argument is only used if object is an instance of the bundle class. When set to NULL, all emulators in the bundle are drawn. Defaults to NULL.

Details

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

Value

A patchwork object.

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

[Updated]

This function builds and trains a GP emulator.

Usage

```
gp(
  X,
  Y,
  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,
  check_rep = TRUE,
  vecchia = FALSE,
  M = 25,
```

```
  ord = 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.
name	kernel function to be used. Either "sexp" for squared exponential kernel or "matern2.5" for Matérn-2.5 kernel. Defaults to "sexp".
lengthscale	initial values of lengthscales in the kernel function. It can be a single numeric value or a vector of length ncol(X): • 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, it is assumed that kernel functions across input dimensions have different lengthscales. Defaults to a vector of 0.1.
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.
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". 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.
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 FALSE for deterministic computer models where the emulator should interpolate the training data points. Set nugget to a larger value and the corresponding bool in nugget_est to TRUE for stochastic emulation 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.
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.
scale	the initial variance value. If scale_est = FALSE, the assigned value is fixed during the training. Defaults to 1.

`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 specification 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 function execution. Defaults to `TRUE`.

`check_rep` **[New]** a bool indicating whether to check for 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 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`.

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

- `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.
- `design()` for sequential designs.
- `update()` to update the GP emulator with new inputs and outputs.
- `alm()`, `mice()`, 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)
```

`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. Supported versions are 3.10, 3.11, and 3.12. If `py_ver = NULL`, the default 'python' version '3.10' 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 trace information will be printed during 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)
```

---

lgp *Linked (D)GP emulator construction*

Description

This function constructs a linked (D)GP emulator for a model chain or network.

Usage

```
lgp(struc, emulators, B = 10, activate = TRUE, verb = TRUE, id = NULL)
```

Arguments

struc	a data frame that defines the connection structure between emulators in the linked system, with the following columns: • From_Emulator: the ID of the emulator providing the output. This ID must match the id slot in the corresponding emulator object (produced by gp() or dgp()) within emulators argument of lgp(), or it should be special value "Global", indicating the global inputs to the model chain or network. The id slot is either automatically generated by gp() or dgp(), or can be manually specified via the id argument in these functions or set with the set_id() function. • To_Emulator: the ID of the emulator receiving the input, also matching the id slot in the corresponding emulator object. • From_Output: a single integer specifying the output dimension of the From_Emulator that is being connected to the input dimension of the To_Emulator specified by To_Input. If From_Emulator is "Global", then From_Output indicates the dimension of the global input passed to the To_Emulator. • To_Input: a single integer specifying the input dimension of the To_Emulator that is receiving the From_Output dimension from the From_Emulator. Each row represents a single one-to-one connection between a specified output dimension of From_Emulator and a corresponding input dimension of To_Emulator. If multiple connections are required between two emulators, each connection should be specified in a separate row.
emulators	a list of emulator objects, each containing an id slot that uniquely identifies it within the linked system. The id slot in each emulator object must match the From_Emulator/To_Emulator columns in struc. If the same emulator is used multiple times within the linked system, the list must contain distinct copies of that emulator, each with a unique ID stored in their id slot. Use the set_id() function to produce copies with different IDs to ensure each instance can be uniquely referenced.
B	the number of imputations used for prediction. Increase the value to refine representation of imputation uncertainty. If the system consists of only GP emulators, B is set to 1 automatically. Defaults to 10.
activate	a bool indicating whether the initialized linked emulator should be activated: • If activate = FALSE, lgp() returns an inactive linked emulator, allowing inspection of its structure using summary(). • If activate = TRUE, lgp() returns an active linked emulator, ready for prediction and validation using predict() and validate(), respectively. Defaults to TRUE.
verb	a bool indicating if the trace information on linked (D)GP emulator construction should be printed during the function call. Defaults to TRUE.
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. Defaults 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 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.
  3. metadata: a data frame providing configuration details for each emulator in the linked system, with following columns:
     - Emulator: the ID of an emulator.
     - Layer: the layer in the linked system where the emulator is positioned. A lower Layer number indicates a position closer to the input, with layer numbering increasing as you move away from the input.
     - Pos_in_Layer: the position of the emulator within its layer. A lower Pos_in_Layer number indicates a position higher up in that layer.
     - Total_Input_Dims: the total number of input dimensions of the emulator.
     - Total_Output_Dims: the total number of output dimensions of the emulator.
  4. struc: The linked system structure, as supplied by struc.

The returned lgp object can be used by

- predict() for linked (D)GP predictions.
- validate() for OOS validation.
- plot() for 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", id = "emulator1")
# emulation of model 2
m2 <- dgp(X2, Y2, depth = 2, name = "matern2.5", id = "emulator2")

struc <- data.frame(From_Emulator = c("Global", "emulator1"),
                    To_Emulator = c("emulator1", "emulator2"),
                    From_Output = c(1, 1),
                    To_Input = c(1, 1))
emulators <- list(m1, m2)

# construct the linked emulator for visual inspection
m_link <- lgp(struc, emulators, activate = FALSE)

# visual inspection
summary(m_link)

# build the linked emulator for prediction
m_link <- lgp(struc, emulators, activate = TRUE)
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, ...)

## S3 method for class 'gp'
mice(
  object,
  x_cand = NULL,
  n_cand = 200,
  batch_size = 1,
  M = 50,
  nugget_s = 1e-06,
  workers = 1,
  limits = NULL,
  int = FALSE,
  ...
)

## S3 method for class 'dgp'
mice(
  object,
  x_cand = NULL,
  n_cand = 200,
  batch_size = 1,
  M = 50,
  nugget_s = 1e-06,
  workers = 1,
  limits = NULL,
  int = FALSE,
  aggregate = NULL,
  ...
)

## S3 method for class 'bundle'
mice(
  object,
  x_cand = NULL,
```

```
  n_cand = 200,
  batch_size = 1,
  M = 50,
  nugget_s = 1e-06,
  workers = 1,
  limits = NULL,
  int = FALSE,
  aggregate = NULL,
  ...
)
```

Arguments

`object`	can be one of the following: • the S3 class `gp`. • the S3 class `dgp`. • the S3 class `bundle`.
`...`	any arguments (with names different from those of arguments used in `mice()`) that are used by `aggregate` can be passed here.
`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 and `aggregate` is not supplied, `x_cand` can also be a list. The list must have a length equal to the number of emulators in `object`, with each element being a matrix representing the candidate set for a corresponding emulator in the bundle. Defaults to `NULL`.
`n_cand`	an integer specifying the size of the candidate set to be generated for selecting the next design point(s). This argument is used only when `x_cand` is `NULL`. Defaults to `200`.
`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`.
`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. This argument is only used when `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 single 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. This argument is only used when `x_cand = NULL`. Defaults to `FALSE`.

`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 aggregate scores at different design points.

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

Details

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

Value

1. If `x_cand` is not `NULL`:
   - When `object` is an instance of the `gp` class, a vector of length `batch_size` is returned, containing the positions (row numbers) of the next design points from `x_cand`.
   - When `object` is an instance of the `dgp` class, a vector of length `batch_size * D` is returned, containing the positions (row numbers) of the next design points from `x_cand` to be added to the DGP emulator.
     - `D` is the number of output dimensions of the DGP emulator if no likelihood layer is included.
     - For a DGP emulator with a `Hetero` or `NegBin` likelihood layer, `D = 2`.
     - For a DGP emulator with a `Categorical` likelihood layer, `D = 1` for binary output or `D = K` for multi-class output with `K` classes.
   - When `object` is an instance of the `bundle` class, a matrix is returned with `batch_size` rows and a column for each emulator in the bundle, containing the positions (row numbers) of the next design points from `x_cand` for individual emulators.
2. If `x_cand` is `NULL`:
   - When `object` is an instance of the `gp` class, a matrix with `batch_size` rows is returned, giving the next design points to be evaluated.
   - When `object` is an instance of the `dgp` class, a matrix with `batch_size * D` rows is returned, where:
     - `D` is the number of output dimensions of the DGP emulator if no likelihood layer is included.
     - For a DGP emulator with a `Hetero` or `NegBin` likelihood layer, `D = 2`.

  - For a DGP emulator with a `Categorical` likelihood layer, `D = 1` for binary output or `D = K` for multi-class output with `K` classes.
- When `object` is an instance of the `bundle` class, a list is returned with a length equal to the number of emulators in the bundle. Each element of the list is a matrix with `batch_size` rows, where each row represents a design point to be added to the corresponding emulator.

Note

The first column of the matrix supplied to the first argument of `aggregate` must correspond to the first output dimension of the DGP emulator if `object` is an instance of the `dgp` class, and so on for subsequent columns and dimensions. If `object` is an instance of the `bundle` class, the first column must correspond to the first emulator in the bundle, and so on for subsequent columns and emulators.

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)
```

`nllik` *Calculate the predictive negative log-likelihood*

Description

This function computes the predictive negative 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 `likelihood` not being `NULL`;
`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.

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
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 iterations and the target RMSE of 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 additional 10 iterations and the same target
m <- design(m, N = 10, limits = lim, f = f, x_test = validate_x, y_test = validate_y, target = 0.01)

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

# inspect the traces of RMSEs of the two packed emulators
draw(m, 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)
```

`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",
  sample_size = 50,
  style = 1,
  min_max = TRUE,
  normalize = 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",
  sample_size = 50,
  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",
  sample_size = 50,
```

```
  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 within the design 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 is an integer referring to the dimension of the global input to the linked emulator system. This argument is only used when style = 1. Defaults to NULL.
method	same as that of validate().
sample_size	same as that of validate().
style	either 1 or 2, indicating two different plotting styles for validation.
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. This argument is not applicable to DGP emulators with categorical likelihoods.
normalize	**[New]** a bool indicating if normalization will be used to scale the counts in validation plots of DGP emulators with categorical likelihoods when style = 2. 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. This argument is not applicable to DGP emulators with categorical likelihoods. Defaults to 'points'
verb	a bool indicating if trace information on plotting will be printed during 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() works well. However, for moderate- to large-scale validation, it is recommended to first run validate() to obtain and store validation results in the emulator object, and then supply the object to plot(). plot() checks the object's loo and oos slots prior to calling validate() and will not perform further calculation if the required information is already stored.
- plot() will only use stored OOS validation if x_test and y_test are identical to those used by validate() to produce the data contained in the object's oos slot, otherwise plot() will re-evaluate OOS validation before plotting.
- The returned patchwork::patchwork object contains the ggplot2::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)
```

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

Description

This function implements prediction 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 must be a matrix representing the global input, where each row corresponds to a test data point and each column represents a global input dimension. The column indices in x must align with the indices specified in the From_Output column of the struc data frame (used in lgp()), corresponding to rows where the From_Emulator column is "Global".
method	**[Updated]** the prediction approach to use: either the mean-variance approach ("mean_var") or the sampling approach ("sampling"). The mean-variance approach returns the means and variances of the predictive distributions, while the sampling approach generates samples from predictive distributions using the derived means and variances. 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 or a 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 prediction. 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.

- **[Updated]** 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). If the likelihood node is categorical, the matrices contain the predictive means and variances of the class probabilities, with columns corresponding to different classes.
  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). If the likelihood node is categorical, the matrices named `layerL` in both `mean` and `var` contain the predictive means and variances of the class probabilities, respectively, with columns corresponding to different classes.
  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`. If the likelihood node in the final layer is categorical, `results` contains D matrices (where D is the number of classes) of sampled class probabilities, each named according to its corresponding class label. Each matrix in `results` 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`. If the likelihood node in the final layer is categorical, `layerL` contains D matrices (where D is the number of classes) of sampled class probabilities, each named according to its corresponding class label. 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()`.
- 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 (same number of emulators in the final layer of the system) matrices named using the `ID`s of the corresponding emulators in the final layer. Each matrix has 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 (same number of emulators in the corresponding layer of the system) matrices named using the IDs of the corresponding emulators in that layer. 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 (same number of emulators in the final layer of the system) sub-lists named using the IDs of the corresponding emulators in the final layer. Each sub-list contains D matrices, named output1, output2,..., outputD, that correspond to the output dimensions of the GP/DGP emulator. Each matrix has 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 (same number of emulators in the corresponding layer of the system) components named using the IDs of the corresponding emulators in that layer. Each component 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.
- the value of sample_size if method = "sampling".

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 static pruning for a DGP emulator.

Usage

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

Arguments

object	an instance of the dgp class that is generated by dgp().
control	a list that can supply the following two components to control static pruning of the DGP emulator: • min_size, the minimum number of design points required to trigger pruning. Defaults to 10 times of the input dimensions. • threshold, the R^2 value above which a GP node is considered redundant and removable. Defaults to 0.97.
verb	a bool indicating if 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 recommended that the design of the DGP emulator is enriched and its structure pruned dynamically using the design() function. Depending on the design of the DGP emulator, static pruning may not be accurate. It is thus recommended that dynamic pruning is implemented as a part of a 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)
```

`serialize` *Serialize the constructed emulator*

Description

This function serializes the constructed emulator.

Usage

```
serialize(object, light = TRUE)
```

Arguments

`object`	an instance of the S3 class `gp`, `dgp`, `lgp`, or `bundle`.
`light`	a bool indicating if a light version of the constructed emulator (that requires a small storage) will be serialized. Defaults to `TRUE`.

Details

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

Value

A serialized version of `object`.

Note

Since the constructed emulators are 'python' objects, they cannot be directly exported to other R processes for parallel processing. This function provides a solution by converting the emulators into serialized objects, which can be restored using `deserialize()` for multi-process parallel implementation.

Examples

```
## Not run:

library(parallel)
library(dgpsi)

# model
f <- function(x) {
 (sin(7.5*x)+1)/2
}

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

# train a DGP emulator
m <- dgp(X, Y, name = "matern2.5")

# testing input data
X_dgp <- seq(0, 1, length = 100)

# serialize the DGP emulator
m_serialized <- serialize(m)

# create a cluster with 8 workers for parallel predictions
cl <- makeCluster(8)

# export objects to the cluster
clusterExport(cl, varlist = c("m_serialized", "X_dgp"))

# initialize deserialized object on each worker
res <- clusterEvalQ(cl, {
  library(dgpsi)
  assign("m_deserialized", deserialize(m_serialized), envir = .GlobalEnv)
})

# perform parallel predictions
results <- parLapply(cl, 1:length(X_dgp), function(i) {
  mean_i <- predict(m_deserialized, X_dgp[i])$results$mean
})

# reset the cluster
stopCluster(cl)

# combine mean predictions
pred_mean <- do.call(rbind, results)

## End(Not run)
```

`set_id` *Set Emulator ID*

Description

This function assigns a unique identifier to an emulator.

Usage

```
set_id(object, id)
```

Arguments

object	an emulator object to which the ID will be assigned.
id	a unique identifier for the emulator as either a numeric or character string. Ensure this ID does not conflict with other emulator IDs, especially when used in linked emulations.

Details

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

Value

The updated object, with the assigned ID stored in its id slot.

Examples

```
## Not run:

# See lgp() 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 prediction 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 improve imputation uncertainty quantification. Decrease the value to improve speed of prediction. 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_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 bool to indicate the addition or removal of the Vecchia approximation: • if `object` is an instance of the `gp` or `dgp` class, `vecchia` indicates either addition (`vecchia = TRUE`) or removal (`vecchia = FALSE`) of the Vecchia approximation from `object`. • if `object` is an instance of the `lgp` class, `vecchia` indicates either addition (`vecchia = TRUE`) or removal (`vecchia = FALSE`) of the Vecchia approximation from all individual (D)GP emulators contained in `object`. Defaults to `TRUE`.
`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 provides a summary of key information for a GP, DGP, or linked (D)GP emulator by generating either a table or an interactive plot of the emulator's structure.

Usage

```
## S3 method for class 'gp'
summary(object, type = "plot", ...)

## S3 method for class 'dgp'
summary(object, type = "plot", ...)

## S3 method for class 'lgp'
summary(object, type = "plot", group_size = 1, ...)
```

Arguments

object	can be one of the following: • the S3 class gp. • the S3 class dgp. • the S3 class lgp.
type	a character string, either "table" or "plot", indicating the format of the output. If set to "table", the function returns a summary in table. If set to "plot", the function returns an interactive visualization. Defaults to "plot".
...	Any arguments that can be passed to kableExtra::kbl() when type = "table".
group_size	an integer specifying the number of consecutive layers to be grouped together in the interactive visualization of linked emulators when type = "plot". This argument is only applicable if object is an instance of the lgp class. Defaults to 1.

Details

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

Value

Either a summary table (returned as kableExtra object) or an interactive visualization (returned as a visNetwork object) of the emulator. The visualization is compatible with R Markdown documents and the RStudio Viewer. The summary table can be further customized by kableExtra::kableExtra package. The resulting visNetwork object can be saved as an HTML file using visNetwork::visSave() from the visNetwork::visNetwork package.

Examples

```
## Not run:

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

## End(Not run)
```

trace_plot *Trace plot for DGP hyperparameters*

Description

This function draws trace plots for the hyperparameters 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 that any further manipulations of unpacked individual emulators will not impact those 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 = TRUE,
  reset = FALSE,
  verb = TRUE,
  N = NULL,
  cores = 1,
  ess_burn = 10,
  B = NULL,
  ...
)

## S3 method for class 'gp'
update(object, X, Y, refit = TRUE, 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 represents 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 TRUE.
reset	a bool indicating whether to reset hyperparameters of the emulator object to the initial values first obtained when the emulator was constructed. Use if it is suspected that a local mode for the hyperparameters has been reached through successive updates. Defaults to FALSE.
verb	a bool indicating if 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 sampler at each I-step of the training of 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.

Examples

```
## Not run:

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

## End(Not run)
```

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

Description

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

Usage

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

## S3 method for class 'gp'
validate(
  object,
  x_test = NULL,
```

```
  y_test = NULL,
  method = "mean_var",
  sample_size = 50,
  verb = TRUE,
  M = 50,
  force = FALSE,
  cores = 1,
  ...
)

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

## S3 method for class 'lgp'
validate(
  object,
  x_test = NULL,
  y_test = NULL,
  method = "mean_var",
  sample_size = 50,
  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`	OOS testing input data: • if `object` is an instance of the gp or dgp class, `x_test` is a matrix where each row is a new input location to be used for validating the emulator and each column is an input dimension.

- if `object` is an instance of the `lgp` class, `x_test` must be a matrix representing the global input, where each row corresponds to a test data point and each column represents a global input dimension. The column indices in `x_test` must align with the indices specified in the `From_Output` column of the `struc` data frame (used in `lgp()`), corresponding to rows where the `From_Emulator` column is `"Global"`.

`x_test` must be provided if `object` is an instance of the `lgp`. `x_test` must also be provided if `y_test` is provided. Defaults to `NULL`, in which case LOO validation is performed.

`y_test` the OOS output data corresponding to `x_test`:

- if `object` is an instance of the `gp` class, `y_test` is a matrix with only one column where each row represents the output corresponding to the matching row of `x_test`.
- if `object` is an instance of the `dgp` class, `y_test` is a matrix where each row represents the output corresponding to the matching row of `x_test` and with columns representing output dimensions.
- if `object` 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 should be 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 L matrices, where L is the number of emulators in the final layer of the system. 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 if `object` is an instance of the `lgp`. `y_test` must also be provided if `x_test` is provided. Defaults to `NULL`, in which case LOO validation is performed.

`method` **[Updated]** the prediction approach to use for validation: either the mean-variance approach (`"mean_var"`) or the sampling approach (`"sampling"`). For details see `predict()`. Defaults to `"mean_var"`.

`sample_size` the number of samples to draw for each given imputation if `method = "sampling"`. Defaults to `50`.

`verb` a bool indicating if trace information for validation should be printed during function execution. Defaults to `TRUE`.

`M` the size of the conditioning set for the Vecchia approximation in 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 LOO or OOS re-evaluation when the `loo` or `oos` slot already exists in `object`. When `force = FALSE`, `validate()` will only re-evaluate the emulators if the `x_test` and `y_test` are not identical to the values in the `oos` slot. If the existing `loo` or `oos` validation used a different `M` in a Vecchia approximation or a different `method` to the one prescribed in this call, the emulator will be re-evaluated. Set `force` to `TRUE` when LOO or OOS re-evaluation is required. Defaults to `FALSE`.

cores	the number of processes to be used for validation. 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 (max-min) normalized root mean/median squared error of the GP emulator. The max-min normalization uses the maximum and minimum values of the validation outputs contained in y_train (or y_test).
  - an integer called M that contains the size of the conditioning set used for the Vecchia approximation, if used, for emulator validation.
  - an integer called sample_size that contains the number of samples used for validation if method = "sampling".

  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 (max-min) normalized root mean/median squared errors of the DGP emulator across different output dimensions. The max-min normalization uses 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, for emulator validation.
  - an integer called sample_size that contains the number of samples used for validation if method = "sampling".

  The rows and columns of matrices (mean, median, std, lower, and upper) correspond to the validation positions and DGP emulator output dimensions, respectively.

- **[Updated]** If object is an instance of the dgp class with a categorical likelihood, 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 vector called label that contains predictive labels from the DGP emulator at validation positions.
  - a matrix called probability that contains mean predictive probabilities for each class from the DGP emulator at validation positions. The matrix has its rows corresponding to validation positions and columns corresponding to different classes.
  - a scalar called log_loss that represents the log loss of the trained DGP classifier. Log loss measures the accuracy of probabilistic predictions, with lower values indicating better classification performance. log_loss ranges from 0 to positive infinity, where a value closer to 0 suggests more confident and accurate predictions.
  - a scalar called accuracy that represents the accuracy of the trained DGP classifier. Accuracy measures the proportion of correctly classified instances among all predictions, with higher values indicating better classification performance. accuracy ranges from 0 to 1, where a value closer to 1 suggests more reliable and precise predictions.
  - a slot named method indicating whether the matrix in the probability slot were obtained using the "mean-var" method or the "sampling" method.
  - an integer called M that contains size of the conditioning set used for the Vecchia approximation, if used, in emulator validation.
  - an integer called sample_size that contains the number of samples used for validation.
- 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 (max-min) normalized root mean/median squared errors of the linked (D)GP emulator. The max-min normalization uses 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 emulator validation.
- an integer called sample_size that contains the number of samples used for validation if method = "sampling".

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, LOO cross validation will be implemented. Otherwise, OOS validation will be implemented. LOO validation is only applicable to a GP or DGP emulator (i.e., object is an instance of the gp or dgp class). If a linked (D)GP emulator (i.e., object is an instance of the lgp class) is provided, x_test and y_test must also be provided for OOS validation.

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, ...)

## S3 method for class 'gp'
vigf(
  object,
  x_cand = NULL,
  n_start = 10,
  batch_size = 1,
  M = 50,
```

```
  workers = 1,
  limits = NULL,
  int = FALSE,
  ...
)

## S3 method for class 'dgp'
vigf(
  object,
  x_cand = NULL,
  n_start = 10,
  batch_size = 1,
  M = 50,
  workers = 1,
  limits = NULL,
  int = FALSE,
  aggregate = NULL,
  ...
)

## S3 method for class 'bundle'
vigf(
  object,
  x_cand = NULL,
  n_start = 10,
  batch_size = 1,
  M = 50,
  workers = 1,
  limits = NULL,
  int = FALSE,
  aggregate = NULL,
  ...
)
```

Arguments

`object`	can be one of the following: • the S3 class `gp`. • the S3 class `dgp`. • the S3 class `bundle`.
`...`	any arguments (with names different from those of arguments used in `vigf()`) that are used by `aggregate` can be passed here.
`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 and `aggregate` is not supplied, `x_cand` can also be a list. The list must have a length equal to the number of emulators in `object`, with each element being a matrix representing the candidate set for a corresponding emulator in the bundle. Defaults to `NULL`.

`n_start` an integer that gives the number of initial design points to be used to determine next design point(s). This argument is only used when `x_cand` is `NULL`. Defaults to `10`.

`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 design point selection. If set to `NULL`, the number of processes is set to `max physical cores available %/% 2`. Defaults to `1`. The argument does not currently support Windows machines when the `aggregate` function is provided, due to the significant overhead caused by initializing the Python environment for each worker under spawning.

`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. This argument is only used when `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 single 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. This argument is only used when `x_cand = NULL`. Defaults to `FALSE`.

`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 aggregate scores at different design points.

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

Details

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

Value

1. If `x_cand` is not `NULL`:

- When `object` is an instance of the `gp` class, a vector of length `batch_size` is returned, containing the positions (row numbers) of the next design points from `x_cand`.
- When `object` is an instance of the `dgp` class, a vector of length `batch_size * D` is returned, containing the positions (row numbers) of the next design points from `x_cand` to be added to the DGP emulator.
  - `D` is the number of output dimensions of the DGP emulator if no likelihood layer is included.
  - For a DGP emulator with a `Hetero` or `NegBin` likelihood layer, `D = 2`.
  - For a DGP emulator with a `Categorical` likelihood layer, `D = 1` for binary output or `D = K` for multi-class output with `K` classes.
- When `object` is an instance of the `bundle` class, a matrix is returned with `batch_size` rows and a column for each emulator in the bundle, containing the positions (row numbers) of the next design points from `x_cand` for individual emulators.

2. If `x_cand` is `NULL`:
   - When `object` is an instance of the `gp` class, a matrix with `batch_size` rows is returned, giving the next design points to be evaluated.
   - When `object` is an instance of the `dgp` class, a matrix with `batch_size * D` rows is returned, where:
     - `D` is the number of output dimensions of the DGP emulator if no likelihood layer is included.
     - For a DGP emulator with a `Hetero` or `NegBin` likelihood layer, `D = 2`.
     - For a DGP emulator with a `Categorical` likelihood layer, `D = 1` for binary output or `D = K` for multi-class output with `K` classes.
   - When `object` is an instance of the `bundle` class, a list is returned with a length equal to the number of emulators in the bundle. Each element of the list is a matrix with `batch_size` rows, where each row represents a design point to be added to the corresponding emulator.

Note

The first column of the matrix supplied to the first argument of `aggregate` must correspond to the first output dimension of the DGP emulator if `object` is an instance of the `dgp` class, and so on for subsequent columns and dimensions. If `object` is an instance of the `bundle` class, the first column must correspond to the first emulator in the bundle, and so on for subsequent columns and emulators.

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)

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

# locate the next design point using VIGF
X_new <- vigf(m, limits = lim)

# 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 sequence of hyperparameter estimates within a DGP emulator*

Description

This function trims the sequence of hyperparameter estimates within a DGP emulator generated during 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 sequence.
end	the last iteration after which all iterations are trimmed from the sequence. 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 sequence. Defaults to 1.

Details

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

Value

An updated object with a trimmed sequence of hyperparameters.

Note

- This function is useful when a DGP emulator has been trained and one wants to trim the sequence of hyperparameters estimated and to use the trimmed sequence to generate point estimates of the DGP model parameters for prediction.
- 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 less disk space to store) will be saved. Defaults to TRUE.

Details

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

Value

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

Note

Since emulators built from the package are 'python' objects, save() from R will not work as it would for R objects. If object was processed by set_vecchia() to add or remove the Vecchia approximation, light should be set to FALSE to ensure reproducibility after the saved emulator is reloaded by read().

Examples

```
## Not run:

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

## End(Not run)
```

Index