Package ‘simlandr’

January 23, 2024

Type Package
Title Simulation-Based Landscape Construction for Dynamical Systems
Version 0.3.1
Description A toolbox for constructing potential landscapes for dynamical systems using Monte Carlo simulation. The method is based on the potential landscape definition by Wang et al. (2008) <doi:10.1073/pnas.0800579105> (also see Zhou & Li, 2016 <doi:10.1063/1.4943096> for further mathematical discussions) and can be used for a large variety of models.
License GPL (>= 3)

BugReports https://github.com/Sciurus365/simlandr/issues

Imports bigmemory, digest, dplyr, forcats, gganimate, ggplot2, grDevices, htmlwidgets, ks, lifecycle, magrittr, MASS, methods, plotly, progress, purrr, rlang, tibble

Suggests coda, knitr, rmarkdown, webshot

Encoding UTF-8

RoxygenNote 7.3.0

NeedsCompilation no

Author Jingmeng Cui [aut, cre] (<https://orcid.org/0000-0003-3421-8457>)

Maintainer Jingmeng Cui <jingmeng.cui@outlook.com>

Repository CRAN

Date/Publication 2024-01-23 18:40:09 UTC

R topics documented:

arg_set-class .................................................. 2
attach_all_matrices ........................................... 4
autolayer.barrier ............................................. 4
arg_set-class

Create and modify argument sets, then make an argument grid for batch simulation

Description

An argument set contains the descriptions of the relevant variables in a batch simulation. Use new_arg_set() to create an arg_set object, and use add_arg_ele() to add an element to the arg_set. After adding all elements in the argument set, use make_arg_grid() to make an argument grid that can be used directly for running batch simulation.

Usage

new_arg_set()

add_arg_ele(arg_set, arg_name, ele_name, start, end, by)

nele(arg_set)

narg(arg_set)

## S3 method for class 'arg_set'
print(x, detail = FALSE, ...)

make_arg_grid(arg_set)
## S3 method for class 'arg_grid'
print(x, detail = FALSE, ...)

### Arguments

- **arg_set**: An `arg_set` object.
- **arg_name, ele_name**: The name of the argument and its element in the simulation function.
- **start, end, by**: The data points where you want to test the variables. Passed to `seq`.
- **x**: An `arg_set` object.
- **detail**: Do you want to print the object details as a full list?
- **...**: Not in use.

### Value

- `new_arg_set()` returns an `arg_set` object.
- `add_arg_ele()` returns an `arg_set` object.
- `nele()` returns an integer.
- `narg()` returns an integer.
- `make_arg_grid()` returns an `arg_grid` object.

### Functions

- `new_arg_set()`: Create an `arg_set`.
- `add_arg_ele()`: Add an element to an `arg_set`.
- `nele()`: The number of elements in an `arg_set`.
- `narg()`: The number of arguments in an `arg_set`.
- `print(arg_set)`: Print an `arg_set` object.
- `make_arg_grid()`: Make an argument grid from an argument set.
- `print(arg_grid)`: Print an `arg_grid` object

### See Also

- `batch_simulation()` for running batch simulation and a concrete example.
attach_all_matrices  
Attach all matrices in a batch simulation

Description
Attach all matrices in a batch simulation

Usage
attach_all_matrices(bs, backingpath = "bp")

Arguments
bs  A batch_simulation object.
backingpath  Passed to bigmemory::as.big.matrix().

Value
A batch_simulation object with all hash_big_matrixes attached.

autolayer.barrier  Get a ggplot2 layer from a barrier object

Description
This layer can show the saddle point (2d) and the minimal energy path (3d) on the landscape.

Usage
## S3 method for class 'barrier'
autolayer(object, path = TRUE, ...)

Arguments
object  A barrier object.
path  Show the minimum energy path in the graph?
...  Not in use.

Value
A ggplot2 layer that can be added to an existing landscape plot.
Perform a batch simulation.

Description
Perform a batch simulation.

Usage
batch_simulation(arg_grid, sim_fun, default_list, bigmemory = TRUE, ...)

## S3 method for class 'batch_simulation'
print(x, detail = FALSE, ...)

Arguments
- **arg_grid**: An arg_grid object. See `make_arg_grid()`.
- **sim_fun**: The simulation function. See `sim_fun_test()` for an example.
- **default_list**: A list of default values for `sim_fun`.
- **bigmemory**: Use `hash_big_matrix-class()` to store large matrices?
- **...**: Other parameters passed to `sim_fun`
- **x**: An arg_set object
- **detail**: Do you want to print the object details as a full list?

Value
A batch_simulation object, also a data frame. The first column, `var`, is a list of `ele_list` that contains all the variables; the second to the second last columns are the values of the variables; the last column is the output of the simulation function.

Functions
- `batch_simulation()`: Perform a batch simulation.

Examples
```r
batch_arg_set_grad <- new_arg_set()
batch_arg_set_grad <- batch_arg_set_grad %>
add_arg_ele(
    arg_name = "parameter", ele_name = "a",
    start = -6, end = -1, by = 1
)
batch_grid_grad <- make_arg_grid(batch_arg_set_grad)
batch_output_grad <- batch_simulation(batch_grid_grad, sim_fun_grad,
default_list = list(
    initial = list(x = 0, y = 0),
    parameter = list(a = -4, b = 0, c = 0, sigmasq = 1)
)```
calculate_barrier

Functions for calculating energy barrier from landscapes

Description

Functions for calculating energy barrier from landscapes

Usage

```r
calculate_barrier(l, ...)
```

## S3 method for class `2d_landscape`
```r
calculate_barrier(
  l,
  start_location_value,
  start_r,
  end_location_value,
  end_r,
  base = exp(1),
  ...
)
```

## S3 method for class `3d_landscape`
```r
calculate_barrier(
  l,
  start_location_value,
  start_r,
  end_location_value,
  end_r,
  Umax,
  expand = TRUE,
  omit_unstable = FALSE,
  base = exp(1),
  ...
)
```

## S3 method for class `2d_landscape_batch`
```r
calculate_barrier(
  l,
  bg = NULL,
  ...)
calculate_barrier

    start_location_value,
    start_r,
    end_location_value,
    end_r,
    base = exp(1),
    ...
)

## S3 method for class '3d_landscape_batch'
calculate_barrier(
    l,
    bg = NULL,
    start_location_value,
    start_r,
    end_location_value,
    end_r,
    Umax,
    expand = TRUE,
    omit_unstable = FALSE,
    base = exp(1),
    ...
)

Arguments

l A landscape object.
...
Not in use.
start_location_value, end_location_value
    The initial position (in value) for searching the start/end point.
start_r, end_r
    The search radius (in L1 distance) for the start/end point.
base
    The base of the log function.
Umax
    The highest possible value of the potential function.
expand
    If the values in the range all equal to Umax, expand the range or not?
omit_unstable
    If a state is not stable (the "local minimum" overlaps with the saddle point), omit
    that state or not?
bg A 2d_barrier_grid or 3d_barrier_grid object if you want to use different
    parameters for each condition. Otherwise NULL as default.

Value

A barrier object that contains the (batch) barrier calculation result(s).
check_conv

---

**check_conv**

*Graphical diagnoses to check if the simulation converges*

**Description**

Compare the distribution of different stages of simulation (for `plot_type == "bin"` or `plot_type = "density"`), or show how the percentiles of the distribution evolve over time (for `plot_type == cumuplot`, see `coda::cumuplot()` for details). More convergence checking methods for MCMC data are available at the coda package. Be cautious: each convergence checking method has its shortcomings, so do not blindly use any results as the definitive conclusion that a simulation converges or not.

**Usage**

```r
check_conv(output, vars, sample_perc = 0.2, plot_type = "bin")
```

### S3 method for class 'check_conv'

```r
print(x, ask = TRUE, ...)
```

**Arguments**

- **output**: A matrix of simulation output.
- **vars**: The names of variables to check.
- **sample_perc**: The percentage of data sample for the initial, middle, and final stage of the simulation. Not required if `plot_type == "cumuplot"`.
- **plot_type**: Which type of plots should be generated? ("bin", "density", or "cumuplot" which uses `coda::cumuplot()`)
- **x**: The object.
- **ask**: Ask to press enter to see the next plot?
- **...**: Not in use.

**Value**

A `check_conv` object that contains the convergence checking result (for `plot_type == "bin"` or `plot_type = "density"`), or draw the cumuplot without a return value (for `plot_type == cumuplot`).

**Methods (by generic)**

- `print(check_conv)`: Print a `checkConv` object.
**get_dist**

*Get the probability distribution from a landscape object*

**Description**
Get the probability distribution from a landscape object

**Usage**

```r
get_dist(l, index = 1)
```

**Arguments**
- `l`: A landscape project.
- `index`: 1 to get the distribution in tidy format; 2 or "raw" to get the raw simulation result (batch_simulation).

**Value**
A data.frame that contains the distribution in the tidy format or the raw simulation result.

---

**hash_big_matrix-class**

*Class “hash_big_matrix”: big matrix with a md5 hash reference*

**Description**
hash_big_matrix class is a modified class from bigmemory::big.matrix-class(). Its purpose is to help users operate big matrices within hard disk in a reusable way, so that the large matrices do not consume too much memory, and the matrices can be reused for the next time. Comparing with bigmemory::big.matrix-class(), the major enhancement of hash_big_matrix class is that the backing files are, by default, stored in a permanent place, with the md5 of the object as the file name. With this explicit name, hash_big_matrix objects can be easily reloaded into workspace every time.

**Usage**

```r
as_hash_big_matrix(x, backingpath = "bp", silence = TRUE, ...)
attach_hash_big_matrix(x, backingpath = "bp")
```

**Arguments**
- `x`: A matrix, vector, or data.frame for bigmemory::as.big.matrix().
- `backingpath`, `...`: Passed to bigmemory::as.big.matrix().
- `silence`: Suppress messages?
Functions

- `as_hash_big_matrix()`: Create a hash_big_matrix object from a matrix.
- `attach_hash_big_matrix()`: Attach a hash_big_matrix object from the backing file to the workspace.

Slots

- `md5`: The md5 value of the matrix.
- `address`: Inherited from `big.matrix`.

---

**make_2d_matrix**

*Make a matrix of 2D static landscape plots for one or two parameters*

**Description**

Make a matrix of 2D static landscape plots for one or two parameters

**Usage**

```r
make_2d_matrix(
  bs,
  x,
  rows = NULL,
  cols,
  lims,
  kde_fun = c("ks", "base"),
  n = 200,
  h,
  adjust = 1,
  Umax = 5,
  individual_landscape = TRUE
)
```

**Arguments**

- `bs`: A batch_simulation object created by `batch_simulation()`.
- `x`: The name of the target variable.
- `rows, cols`: The names of the parameters. `rows` can be left blank if only one parameter is needed.
- `lims`: The limits of the range for the density estimator as `c(x1, xu)` for 2D landscapes, `c(x1, xu, y1, yu)` for 3D landscapes, `c(x1, xu, y1, yu, z1, zu)` for 4D landscapes. If missing, the range of the data extended by 10% for both sides will be used. For landscapes based on multiple simulations, the largest range of all simulations (which means the lowest lower limit and the highest upper limit) will be used by default.
**make_2d_static**

Which kernel estimator to use? Choices: "ks" `ks::kde()` (default; faster and using less memory); "base" `base::density()` (only for 2D landscapes); "MASS" `MASS::kde2d()` (only for 3D landscapes).

n
The number of equally spaced points in each axis, at which the density is to be estimated.

h
A number, or possibly a vector for 3D and 4D landscapes, specifying the smoothing bandwidth to be used. If missing, the default value of the kernel estimator will be used (but `bw = "SJ"` for `base::density()`). Note that the definition of bandwidth might be different for different kernel estimators. For landscapes based on multiple simulations, the largest h of all simulations will be used by default.

adjust
The multiplier to the bandwidth. The bandwidth used is actually `adjust * h`. This makes it easy to specify values like "half the default" bandwidth.

Umax
The maximum displayed value of potential.

individual_landscape
Make individual landscape for each simulation? Default is `TRUE` so that it is possible to calculate barriers. Set to `FALSE` to save time.

**Value**

A `2d_matrix_landscape` object that describes the landscape of the system, including the smoothed distribution and the landscape plot.

---

**Usage**

```
make_2d_static(
  output,  
x,  
lims,  
kde_fun = c("ks", "base"),  
n = 200,  
h,  
adjust = 1,  
Umax = 5)
```

```
make_2d_single(
  output,  
x,  
```

**Description**

Make 2D static landscape plot for a single simulation output
make_3d_animation

```r
make_3d_animation(output, x, lims, kde_fun = c("ks", "base"), n = 200, h, adjust = 1, Umax = 5)
```

**Arguments**

- `output`: A matrix of simulation output.
- `x`: The name of the target variable.
- `lims`: The limits of the range for the density estimator as `c(xl, xu)` for 2D landscapes, `c(xl, xu, yl, yu)` for 3D landscapes, `c(xl, xu, yl, yu, zl, zu)` for 4D landscapes. If missing, the range of the data extended by 10% for both sides will be used. For landscapes based on multiple simulations, the largest range of all simulations (which means the lowest lower limit and the highest upper limit) will be used by default.
- `kde_fun`: Which kernel estimator to use? Choices: "ks" `ks::kde()` (default; faster and using less memory); "base" `base::density()` (only for 2D landscapes); "MASS" `MASS::kde2d()` (only for 3D landscapes).
- `n`: The number of equally spaced points in each axis, at which the density is to be estimated.
- `h`: A number, or possibly a vector for 3D and 4D landscapes, specifying the smoothing bandwidth to be used. If missing, the default value of the kernel estimator will be used (but `bw = "SJ"` for `base::density()`). Note that the definition of bandwidth might be different for different kernel estimators. For landscapes based on multiple simulations, the largest `h` of all simulations will be used by default.
- `adjust`: The multiplier to the bandwidth. The bandwidth used is actually `adjust * h`. This makes it easy to specify values like "half the default" bandwidth.
- `Umax`: The maximum displayed value of potential.

**Value**

A `2d_static_landscape` object that describes the landscape of the system, including the smooth distribution and the landscape plot.

---

**Description**

Make 3d animations from multiple simulations
Usage

```r
make_3d_animation(
  bs,
  x,
  y,
  fr,
  lims,
  kde_fun = c("ks", "MASS"),
  n = 200,
  h,
  adjust = 1,
  Umax = 5,
  individual_landscape = TRUE,
  mat_3d = FALSE
)
```

Arguments

- **bs**  
  A `batch_simulation` object created by `[batch_simulation()]`.  
- **x, y**  
  The names of the target variables.  
- **fr**  
  The names of the parameters used to represent frames in the animation.  
- **lims**  
  The limits of the range for the density estimator as `c(xl, xu)` for 2D landscapes, `c(xl, xu, yl, yu)` for 3D landscapes, `c(xl, xu, yl, yu, zl, zu)` for 4D landscapes. If missing, the range of the data extended by 10% for both sides will be used. For landscapes based on multiple simulations, the largest range of all simulations (which means the lowest lower limit and the highest upper limit) will be used by default.  
- **kde_fun**  
  Which kernel estimator to use? Choices: "ks" for `ks::kde()` (default; faster and using less memory); "base" for `base::density()` (only for 2D landscapes); "MASS" for `MASS::kde2d()` (only for 3D landscapes).  
- **n**  
  The number of equally spaced points in each axis, at which the density is to be estimated.  
- **h**  
  A number, or possibly a vector for 3D and 4D landscapes, specifying the smoothing bandwidth to be used. If missing, the default value of the kernel estimator will be used (but `bw = "SJ"` for `base::density()`). Note that the definition of bandwidth might be different for different kernel estimators. For landscapes based on multiple simulations, the largest `h` of all simulations will be used by default.  
- **adjust**  
  The multiplier to the bandwidth. The bandwidth used is actually `adjust * h`. This makes it easy to specify values like "half the default" bandwidth.  
- **Umax**  
  The maximum displayed value of potential.  
- **individual_landscape**  
  Make individual landscape for each simulation? Default is `TRUE` so that it is possible to calculate barriers. Set to `FALSE` to save time.  
- **mat_3d**  
  Also make the matrix by `make_3d_matrix()`? If so, the matrix can be drawn with `plot(<landscape>, 3)`.
**Value**

A 3d_animation_landscape object that describes the landscape of the system, including the smoothed distribution and the landscape plot.

---

**make_3d_matrix**  
*Make a matrix of 3D static landscape plots for one or two parameters*

**Description**

Currently only 3D (x, y, color) is supported. Matrices with 3D (x, y, z) plots are not supported.

**Usage**

```r
make_3d_matrix(
  bs,
  x, y,
  rows = NULL,
  cols,
  lims,
  kde_fun = c("ks", "MASS"),
  n = 200,
  h,
  adjust = 1,
  Umax = 5,
  individual_landscape = TRUE
)
```

**Arguments**

- `bs`  
  A batch_simulation object created by `[batch_simulation()]`.
- `x, y`  
  The names of the target variables.
- `rows, cols`  
  The names of the parameters. `rows` can be left blank if only one parameter is needed.
- `lims`  
  The limits of the range for the density estimator as c(xl, xu) for 2D landscapes, c(xl, xu, yl, yu) for 3D landscapes, c(xl, xu, yl, yu, zl, zu) for 4D landscapes. If missing, the range of the data extended by 10% for both sides will be used. For landscapes based on multiple simulations, the largest range of all simulations (which means the lowest lower limit and the highest upper limit) will be used by default.
- `kde_fun`  
  Which kernel estimator to use? Choices: "ks" `ks::kde()` (default; faster and using less memory); "base` base::density()` (only for 2D landscapes); "MASS" `MASS::kde2d()` (only for 3D landscapes).
- `n`  
  The number of equally spaced points in each axis, at which the density is to be estimated.
**make_3d_static**

A number, or possibly a vector for 3D and 4D landscapes, specifying the smoothing bandwidth to be used. If missing, the default value of the kernel estimator will be used (but bw = "SJ" for base::density()). Note that the definition of bandwidth might be different for different kernel estimators. For landscapes based on multiple simulations, the largest h of all simulations will be used by default.

**adjust**

The multiplier to the bandwidth. The bandwidth used is actually adjust * h. This makes it easy to specify values like "half the default" bandwidth.

**Umax**

The maximum displayed value of potential.

**individual_landscape**

Make individual landscape for each simulation? Default is TRUE so that it is possible to calculate barriers. Set to FALSE to save time.

**Value**

A 3d_matrix_landscape object that describes the landscape of the system, including the smoothed distribution and the landscape plot.

---

**Description**

Make 3D static landscape plots from simulation output

**Usage**

```r
make_3d_static(
  output,
  x,
  y,
  lims,
  kde_fun = c("ks", "MASS"),
  n = 200,
  h,
  adjust = 1,
  Umax = 5
)

make_3d_single(
  output,
  x,
  y,
  lims,
  kde_fun = c("ks", "MASS"),
  n = 200,
)```
Arguments

output  A matrix of simulation output.
x, y  The names of the target variables.
lims  The limits of the range for the density estimator as $c(x_l, x_u)$ for 2D landscapes, $c(x_l, x_u, y_l, y_u)$ for 3D landscapes, $c(x_l, x_u, y_l, y_u, z_l, z_u)$ for 4D landscapes. If missing, the range of the data extended by 10% for both sides will be used. For landscapes based on multiple simulations, the largest range of all simulations (which means the lowest lower limit and the highest upper limit) will be used by default.
kde_fun  Which kernel estimator to use? Choices: "ks" $\text{ks::kde()}$ (default; faster and using less memory); "base" $\text{base::density()}$ (only for 2D landscapes); "MASS" $\text{MASS::kde2d()}$ (only for 3D landscapes).
n  The number of equally spaced points in each axis, at which the density is to be estimated.
h  A number, or possibly a vector for 3D and 4D landscapes, specifying the smoothing bandwidth to be used. If missing, the default value of the kernel estimator will be used (but $\text{bw = "SJ"}$ for $\text{base::density()}$). Note that the definition of bandwidth might be different for different kernel estimators. For landscapes based on multiple simulations, the largest $h$ of all simulations will be used by default.
adjust  The multiplier to the bandwidth. The bandwidth used is actually $\text{adjust} \times h$. This makes it easy to specify values like "half the default" bandwidth.
$U_{max}$  The maximum displayed value of potential.

Value

A $\text{3d_static_landscape}$ object that describes the landscape of the system, including the smooth distribution and the landscape plot.

make_4d_static

Make 4D static space-color plots from simulation output

Description

Make 4D static space-color plots from simulation output
make_4d_static

Usage

make_4d_static(
  output,
  x,
  y,
  z,
  lims,
  kde_fun = "ks",
  n = 50,
  h,
  adjust = 1,
  Umax = 5
)

make_4d_single(
  output,
  x,
  y,
  z,
  lims,
  kde_fun = "ks",
  n = 50,
  h,
  adjust = 1,
  Umax = 5
)

Arguments

output A matrix of simulation output.
x, y, z The names of the target variables.
lims The limits of the range for the density estimator as c(xl, xu) for 2D landscapes, c(xl, xu, yl, yu) for 3D landscapes, c(xl, xu, yl, yu,zl,zu) for 4D landscapes. If missing, the range of the data extended by 10% for both sides will be used. For landscapes based on multiple simulations, the largest range of all simulations (which means the lowest lower limit and the highest upper limit) will be used by default.
kde_fun Which kernel estimator to use? Choices: "ks" ks::kde() (default; faster and using less memory); "base" base::density() (only for 2D landscapes); "MASS" MASS::kde2d() (only for 3D landscapes).
n The number of equally spaced points in each axis, at which the density is to be estimated.
h A number, or possibly a vector for 3D and 4D landscapes, specifying the smoothing bandwidth to be used. If missing, the default value of the kernel estimator will be used (but bw = "SJ" for base::density()). Note that the definition of bandwidth might be different for different kernel estimators. For landscapes
based on multiple simulations, the largest $h$ of all simulations will be used by default.

**adjust**
The multiplier to the bandwidth. The bandwidth used is actually $\text{adjust} \times h$. This makes it easy to specify values like "half the default" bandwidth.

**Umax**
The maximum displayed value of potential.

**Value**
A 4d\_static\_landscape object that describes the landscape of the system, including the smoothed distribution and the landscape plot.

---

**make\_barrier\_grid\_2d**  
*Make a grid for calculating barriers for 2d landscapes*

**Description**
Make a grid for calculating barriers for 2d landscapes

**Usage**

```r
make_barrier_grid_2d(
  ag,
  start_location_value, start_r,
  end_location_value, end_r,
  df = NULL,
  print_template = FALSE
)
```

**Arguments**

- **ag**: An arg\_grid object.
- **start_location_value, start_r**, **end_location_value, end_r**: Default values for finding local minimum. See calculate\_barrier().
- **df**: A data frame for the variables. Use print\_template = TRUE to get a template.
- **print_template**: Print a template for df.

**Value**
A barrier\_grid\_2d object that specifies the condition for each barrier calculation.
make_barrier_grid_3d

Make a grid for calculating barriers for 3d landscapes

Description

Make a grid for calculating barriers for 3d landscapes

Usage

```r
make_barrier_grid_3d(
  ag,
  start_location_value, start_r, 
  end_location_value, end_r,
  df = NULL,
  print_template = FALSE
)
```

Arguments

- **ag**: An arg_grid object.
- **start_location_value, start_r, end_location_value, end_r**: Default values for finding local minimum. See `calculate_barrier()`.
- **df**: A data frame for the variables. Use `print_template = TRUE` to get a template.
- **print_template**: Print a template for `df`.

Value

A `barrier_grid_3d` object that specifies the condition for each barrier calculation.

plot.landscape

Make plots from landscape objects

Description

Make plots from landscape objects

Usage

```r
## S3 method for class 'landscape'
plot(x, index = 1, ...)
```
Arguments

- `x`: A landscape object
- `index`: Default is 1. For some landscape objects, there is a second plot (usually 2d heatmaps for 3d landscapes) or a third plot (usually 3d matrices for 3d animations). Use `index = 2` to plot that one.
- `...`: Not in use.

Value

The plot.

save_landscape

Description

Save landscape plots

Usage

```
save_landscape(l, path = NULL, selfcontained = FALSE, ...)
```

Arguments

- `l`: A landscape object
- `path`: The path to save the output. Default: "/pics/x_y.html".
- `selfcontained`: For 'plotly' plots, save the output as a self-contained html file? Default: FALSE.
- `...`: Other parameters passed to `htmlwidgets::saveWidget()` or `ggplot2::ggsave()`

Value

The function saves the plot to a specific path. It does not have a return value.
Description

This is a toy stochastic gradient system which can have bistability in some conditions. Model specification:

\[ U = x^4 + y^4 + axy + bx + cy \]

\[
dx/dt = -\partial U/\partial x + \sigma dW/dt = -4x^3 - ay - b + \sigma dW/dt
\]

\[
dy/dt = -\partial U/\partial y + \sigma dW/dt = -4y^3 - ax - c + \sigma dW/dt
\]

Usage

```r
sim_fun_grad(
  initial = list(x = 0, y = 0),
  parameter = list(a = -4, b = 0, c = 0, sigmasq = 1),
  length = 1e+05,
  stepsize = 0.01,
  seed = NULL
)
```

Arguments

- `initial`, `parameter`  
  Two sets of parameters. `initial` contains the initial value of `x` and `y`; `parameter` contains `a`, `b`, `c`, which control the shape of the potential landscape, and `sigmasq`, which is the square of `\sigma` and controls the amplitude of noise.

- `length`  
  The length of simulation.

- `stepsize`  
  The step size used in the Euler method.

- `seed`  
  The initial seed that will be passed to `set.seed()` function.

Value

A matrix of simulation results.

See Also

`sim_fun_nongrad()` and `batch_simulation()`.
sim_fun_nongrad

**A simple non-gradient simulation function for testing**

**Description**

This is a toy stochastic non-gradient system which can have multistability in some conditions.

**Model specification:**

**Usage**

```r
sim_fun_nongrad(
    initial = list(x1 = 0, x2 = 0, a = 1),
    parameter = list(b = 1, k = 1, S = 0.5, n = 4, lambda = 0.01, sigmasq1 = 8, sigmasq2 = 8, sigmasq3 = 2),
    constrain_a = TRUE,
    amin = -0.3,
    amax = 1.8,
    length = 1e+05,
    stepsize = 0.01,
    seed = NULL,
    progress = TRUE
)
```

**Arguments**

- `initial, parameter` Two sets of parameters. `initial` contains the initial value of `x1`, `x2`, and `a`; `parameter` contains `b, k, S, n, lambda`, which control the model dynamics, and `sigmasq1, sigmasq2, sigmasq3`, which are the squares of `σ_1, σ_2, σ_3` and controls the amplitude of noise.
- `constrain_a` Should the value of `a` be constrained? (TRUE by default).
- `amin, amax` If `constrain_a`, the minimum and maximum values of `a`.
- `length` The length of simulation.
- `stepsize` The step size used in the Euler method.
- `seed` The initial seed that will be passed to `set.seed()` function.
- `progress` Show progress bar of the simulation?

**Details**

\[
\frac{dx_1}{dt} = \frac{ax_1^n}{S^n + x_1^n} + \frac{bS^n}{S^n + x_2^n} - kx_1 + \sigma_1 dW_1/dt \\
\frac{dx_2}{dt} = \frac{ax_2^n}{S^n + x_2^n} + \frac{bS^n}{S^n + x_1^n} - kx_2 + \sigma_2 dW_2/dt \\
\frac{da}{dt} = -\lambda a + \sigma_3 dW_3/dt
\]
**sim_fun_test**

**Value**

A matrix of simulation results.

**References**


**See Also**

`sim_fun_grad()` and `batch_simulation()`.

---

**sim_fun_test**

A simple simulation function for testing

**Description**

A simple simulation function for testing

**Usage**

```r
sim_fun_test(par1, par2, length = 1000)
```

**Arguments**

- `par1, par2` Two parameters. `par1` contains `var1`; `par2` contains `var2` and `var3`.
- `length` The length of simulation.

**Value**

A matrix of simulation results.

**See Also**

`sim_fun_grad()` and `sim_fun_nongrad()` for more realistic examples.
Summary the barrier height from a barrier object

Description

Summarize the barrier height from a barrier object

Usage

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

Arguments

- `object` A barrier object.
- `...` Not in use.

Value

A vector (for a single barrier calculation result) or a dataframe (for batch barrier calculation results) that contains the barrier heights on the landscape.
Index

add_arg_ele (arg_set-class), 2
arg_set-class, 2
as_hash_big_matrix
  (hash_big_matrix-class), 9
attach_all_matrices, 4
attach_hash_big_matrix
  (hash_big_matrix-class), 9
autolayer.barrier, 4
batch_simulation, 5
batch_simulation(), 3, 21, 23
bigmemory::as.big.matrix(), 4, 9
calculate_barrier, 6
calculate_barrier(), 18, 19
check_conv, 8
coda::cumuplot(), 8
get_dist, 9
ggplot2::ggsave(), 20
hash_big_matrix
  (hash_big_matrix-class), 9
hash_big_matrix-class, 9
htmlwidgets::saveWidget(), 20
ks::kde(), 11–14, 16, 17
make_2d_matrix, 10
make_2d_single (make_2d_static), 11
make_2d_static, 11
make_3d_animation, 12
make_3d_matrix, 14
make_3d_matrix(), 13
make_3d_single (make_3d_static), 15
make_3d_static, 15
make_4d_single (make_4d_static), 16
make_4d_static, 16
make_arg_grid (arg_set-class), 2
make_arg_grid(), 5
make_barrier_grid_2d, 18
make_barrier_grid_3d, 19
MASS::kde2d(), 11–14, 16, 17
narg (arg_set-class), 2
nele (arg_set-class), 2
new_arg_set (arg_set-class), 2
plot.landscape, 19
print.arg_grid (arg_set-class), 2
print.arg_set (arg_set-class), 2
print.batch_simulation
  (batch_simulation), 5
print.check_conv (check_conv), 8
save_landscape, 20
sim_fun_grad, 21
sim_fun_grad(), 23
sim_fun_nongrad, 22
sim_fun_nongrad(), 21, 23
sim_fun_test, 23
sim_fun_test(), 5
summary.barrier, 24

25