Package ‘simlandr’

August 24, 2022

Type Package

Title Simulation-Based Landscape Construction for Dynamical Systems

Version 0.2.1

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)

Encoding UTF-8

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

RoxygenNote 7.2.1


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

Suggests knitr, rmarkdown, webshot

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 2022-08-24 07:00:02 UTC

R topics documented:

attach_all_matrices .......................................................... 3
calculate_barrier ............................................................... 3
**R topics documented:**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>calculate_barrier_2d</td>
<td>4</td>
</tr>
<tr>
<td>calculate_barrier_2d_batch</td>
<td>5</td>
</tr>
<tr>
<td>calculate_barrier_3d</td>
<td>6</td>
</tr>
<tr>
<td>calculate_barrier_3d_batch</td>
<td>7</td>
</tr>
<tr>
<td>check_conv</td>
<td>8</td>
</tr>
<tr>
<td>fill_in_struct</td>
<td>8</td>
</tr>
<tr>
<td>find_local_min_2d</td>
<td>9</td>
</tr>
<tr>
<td>find_local_min_3d</td>
<td>9</td>
</tr>
<tr>
<td>get_barrier_height</td>
<td>10</td>
</tr>
<tr>
<td>get_dist</td>
<td>10</td>
</tr>
<tr>
<td>get_geom</td>
<td>11</td>
</tr>
<tr>
<td>hash_big.matrix-class</td>
<td>11</td>
</tr>
<tr>
<td>make_2d_density</td>
<td>12</td>
</tr>
<tr>
<td>make_2d_kernel_dist</td>
<td>13</td>
</tr>
<tr>
<td>make_2d_matrix</td>
<td>13</td>
</tr>
<tr>
<td>make_2d_static</td>
<td>14</td>
</tr>
<tr>
<td>make_2d_tidy_dist</td>
<td>15</td>
</tr>
<tr>
<td>make_3d_animation</td>
<td>15</td>
</tr>
<tr>
<td>make_3d_kernel_dist</td>
<td>16</td>
</tr>
<tr>
<td>make_3d_matrix</td>
<td>17</td>
</tr>
<tr>
<td>make_3d_static</td>
<td>18</td>
</tr>
<tr>
<td>make_3d_tidy_dist</td>
<td>18</td>
</tr>
<tr>
<td>make_4d_static</td>
<td>19</td>
</tr>
<tr>
<td>make_arg_grid</td>
<td>20</td>
</tr>
<tr>
<td>make_barrier_grid_2d</td>
<td>20</td>
</tr>
<tr>
<td>make_barrier_grid_3d</td>
<td>21</td>
</tr>
<tr>
<td>make_var_grid</td>
<td>22</td>
</tr>
<tr>
<td>modified_simulation</td>
<td>22</td>
</tr>
<tr>
<td>narg</td>
<td>23</td>
</tr>
<tr>
<td>nele</td>
<td>24</td>
</tr>
<tr>
<td>new_arg_set</td>
<td>24</td>
</tr>
<tr>
<td>new_var_set</td>
<td>25</td>
</tr>
<tr>
<td>npar</td>
<td>26</td>
</tr>
<tr>
<td>nvar</td>
<td>26</td>
</tr>
<tr>
<td>plot.barrier</td>
<td>27</td>
</tr>
<tr>
<td>plot.landscape</td>
<td>27</td>
</tr>
<tr>
<td>print.arg_grid</td>
<td>28</td>
</tr>
<tr>
<td>print.arg_set</td>
<td>28</td>
</tr>
<tr>
<td>print.batch_simulation</td>
<td>29</td>
</tr>
<tr>
<td>print.check_conv</td>
<td>29</td>
</tr>
<tr>
<td>print.var_grid</td>
<td>30</td>
</tr>
<tr>
<td>print.var_set</td>
<td>30</td>
</tr>
<tr>
<td>reverselog_trans</td>
<td>31</td>
</tr>
<tr>
<td>save_landscape</td>
<td>31</td>
</tr>
<tr>
<td>sim_fun_grad</td>
<td>32</td>
</tr>
<tr>
<td>sim_fun_nongrad</td>
<td>33</td>
</tr>
<tr>
<td>sim_fun_test</td>
<td>34</td>
</tr>
</tbody>
</table>

**Index** 35
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.

calculate_barrier

General function for calculating energy barrier

Description

General function for calculating energy barrier

Usage

calculate_barrier(l, ...)

## S3 method for class '2d_static_landscape'
calculate_barrier(l, ...)

## S3 method for class '2d_density_landscape'
calculate_barrier(l, ...)

## S3 method for class 'density'
calculate_barrier(l, ...)

## S3 method for class '2d_static_landscape'
calculate_barrier(l, ...)

## S3 method for class '3d_static_landscape'
calculate_barrier(l, ...)
calculate_barrier_2d

## S3 method for class 'list'
calculate_barrier(l, ...)

## S3 method for class '3d_animation_landscape'
calculate_barrier(l, ...)

## S3 method for class '3d_matrix_landscape'
calculate_barrier(l, ...)

## S3 method for class '2d_matrix_landscape'
calculate_barrier(l, ...)

### Arguments

- `l`: A landscape or related project.
- `...`: Other parameters.

### Value

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

### See Also

`calculate_barrier_2d()`, `calculate_barrier_2d_batch()`, `calculate_barrier_3d()`, `calculate_barrier_3d_batch()`, `plot.barrier()`

---

**calculate_barrier_2d**  
*Calculate barrier from a 2D landscape*

**Description**

Calculate barrier from a 2D landscape

**Usage**

```r
calculate_barrier_2d(  
  l,  
  start_location_value = 0,  
  start_r = 0.1,  
  end_location_value = 0.7,  
  end_r = 0.15,  
  base = exp(1)  
)
```
calculate_barrier_2d_batch

Arguments

1. A 2d_static_landscape object (recommended) or a density distribution.
2. start_location_value, end_location_value
   The initial position (in value) for searching the start/end point.
3. start_r, end_r
   The searching radius for searching the start/end point.
4. base
   The base of the log function.

Value

A barrier_2d object that contains the barrier calculation result.

calculate_barrier_2d_batch

*Calculate barrier from a 2D landscape with multiple simulations*

Description

Calculate barrier from a 2D landscape with multiple simulations

Usage

```r
calculate_barrier_2d_batch(
    l,
    bg = NULL,
    start_location_value = 0,
    start_r = 0.1,
    end_location_value = 0.7,
    end_r = 0.15,
    base = exp(1)
)
```

Arguments

1. l
   A 2d_animation_landscape (not implemented yet) or a 2d_matrix_landscape.
2. bg
   A barrier_grid_3d object if you want to use different parameters for each condition. Otherwise NULL.
3. start_location_value, end_location_value
   The initial position (in value) for searching the start/end point.
4. start_r, end_r
   The searching (L1) radius for searching the start/end point.
5. base
   The base of the log function.

Value

A barrier_2d_batch object that contains the batch barrier calculation results.
**calculate_barrier_3d**  *Calculate barrier from a 3D landscape*

**Description**

Calculate barrier from a 3D landscape

**Usage**

```r
calculate_barrier_3d(
  l,
  start_location_value = c(0, 0),
  start_r = 0.1,
  end_location_value = c(0.7, 0.6),
  end_r = 0.15,
  Umax,
  expand = TRUE,
  omit_unstable = FALSE,
  base = exp(1)
)
```

**Arguments**

- `l` A 3d_static_landscape object (recommended) or a kde2d distribution.
- `start_location_value, end_location_value` The initial position (in value) for searching the start/end point.
- `start_r, end_r` The searching (L1) radius for searching the start/end point.
- `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?
- `base` The base of the log function.

**Value**

A barrier_3d object that contains the barrier calculation result.
calculate_barrier_3d_batch

Calculate barrier from a 3D landscape with multiple simulations

Description

Calculate barrier from a 3D landscape with multiple simulations

Usage

```r
calculate_barrier_3d_batch(
  l,
  bg = NULL,
  start_location_value = c(0, 0),
  start_r = 0.1,
  end_location_value = c(0.7, 0.6),
  end_r = 0.15,
  Umax,
  expand = TRUE,
  omit_unstable = FALSE,
  base = exp(1)
)
```

Arguments

- `l`: A 3d_animation_landscape or a 3d_matrix_landscape.
- `bg`: A barrier_grid_3d object if you want to use different parameters for each condition. Otherwise NULL.
- `start_location_value`, `end_location_value`: The initial position (in value) for searching the start/end point.
- `start_r`, `end_r`: The searching (L1) radius for searching the start/end point.
- `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?
- `base`: The base of the log function.

Value

A barrier_3d_batch object that contains the batch barrier calculation results.
check_conv

Check density convergence of simulation

Description
Check density convergence of simulation

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

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.
- plot_type: Which type of plots should be generated? ("bin" or "density")

Value
A check_conv object that contains the convergence checking result.

fill_in_struct

Fill a vector of values into a structure list.

Description
Fill a vector of values into a structure list.

Usage
fill_in_struct(vec, struct)

Arguments
- vec: A vector of values.
- struct: A list with a certain structure.

Value
A ele_list object.

See Also
modified_simulation()
**find_local_min_2d**  
*Find local minimum of a 2d distribution*

**Description**
Find local minimum of a 2d distribution

**Usage**
```
fund_local_min_2d(dist, localmin, r)
```

**Arguments**
- **dist**: An density distribution object.
- **localmin**: Starting value of finding local minimum.
- **r**: Searching radius.

**Value**
A list with two elements: **U**, the potential value of the local minimum, and **location**, the position of the local minimum.

---

**find_local_min_3d**  
*Find local minimum of a 3d distribution*

**Description**
Find local minimum of a 3d distribution

**Usage**
```
fund_local_min_3d(dist, localmin, r, Umax, expand = TRUE, first_called = TRUE)
```

**Arguments**
- **dist**: An kde2d distribution object.
- **localmin**: Starting value of finding local minimum.
- **r**: Searching (L1) radius.
- **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?
- **first_called**: Is this function first called by another function?

**Value**
A list with two elements: **U**, the potential value of the local minimum, and **location**, the position of the local minimum.
get_barrier_height  Get the barrier height from a barrier object.

Description

Get the barrier height from a barrier object.

Usage

get_barrier_height(b)

Arguments

b  A barrier object.

Value

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

gget_dist  Get the probability distribution from a landscape object

Description

Get the probability distribution from a landscape object.

Usage

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.
get_geom

Get a ggplot2 geom layer that can be added to a ggplot2 landscape plot

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

Usage
get_geom(b, path = TRUE)

Arguments
b A barrier object.
path Show the minimum energy path in the graph?

Value
A ggplot2 geom (formally a LayerInstance object) that can be added to an existing ggplot.

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
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_density**

*Make 2D density-based landscape plot for a single simulation output*

Description

[Deprecated]

This function was deprecated. Use `make_2d_static()` instead.

Usage

```r
make_2d_density(output, x, adjust = 50, from = -0.1, to = 1, Umax = 5)
```

Arguments

- `output` A matrix of simulation output.
- `x` The name of the target variable.
- `adjust`, `from`, `to` Passed to `density`.
- `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.
**make_2d_kernel_dist**    
Make 2D kernel smooth distribution

**Description**

Make 2D kernel smooth distribution

**Usage**

```r
make_2d_kernel_dist(
  output,
  x,
  y,
  n = 200,
  lims = c(-0.1, 1.1, -0.1, 1.1),
  h,
  kde_fun = "ks"
)
```

**Arguments**

- `output`: A matrix of simulation output.
- `x, y`: The name of the target variable.
- `n, lims, h`: Passed to `ks::kde()` or `MASS::kde2d()`. If using `ks::kde`, `H = diag(h, 2, 2)`. Note: the definition of bandwidth (h) is different in two functions. To get a similar output, the h is about 50 to 5000 times smaller for `ks::kde()` than `MASS::kde2d()`.
- `kde_fun`: Which to use? Choices: "ks" `ks::kde` (default; faster and taking less memory); "MASS" `MASS::kde2d`.

**Value**

A kde2d-type list of smooth distribution.

**make_2d_matrix**    
Make a matrix of 2d graphs for two parameters

**Description**

Make a matrix of 2d graphs for two parameters
Usage

```r
make_2d_matrix(
  bs,
  x,
  rows = NULL,
  cols,
  adjust = 50,
  from = -0.1,
  to = 1,
  Umax = 5,
  individual_landscape = FALSE
)
```

Arguments

- **bs**: A `batch_simulation` object created by `batch_simulation`.
- **x, rows, cols**: The names of the target variables. If `rows` is `NULL`, only a vector of graphs will be generated.
- **adjust, from, to**: Passed to `density`.
- **Umax**: The maximum displayed value of potential.
- **individual_landscape**: Make individual landscape for each simulation?

Value

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

Description

Make 2D static landscape plot for a single simulation output

Usage

```r
make_2d_static(output, x, adjust = 50, from = -0.1, to = 1, Umax = 5)
```

Arguments

- **output**: A matrix of simulation output.
- **x**: The name of the target variable.
- **adjust, from, to**: Passed to `density`.
- **Umax**: The maximum displayed value of potential.
make_2d_tidy_dist

Value

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

Description

Make a tidy data.frame from smooth 2d distribution matrix

Usage

make_2d_tidy_dist(dist_2d, value = NULL, var_name = NULL)

Arguments

dist_2d kde2d distribution.
value The value of the variable of interest.
var_name The name of the variable.

Value

A tidy data.frame.

make_3d_animation

Description

Make 3d animations from multiple simulations

Usage

make_3d_animation(
    bs,
    x,
    y,
    fr,
    Umax = 5,
    n = 200,
    lims = c(-0.1, 1.1, -0.1, 1.1),
    h = 0.001,
    kde_fun = "ks",
    individual_landscape = FALSE,
    mat_3d = TRUE
)
make_3d_kernel_dist

Arguments

- **bs**: A batch_simulation object created by [batch_simulation].
- **x, y, fr**: The names of the target variables. fr corresponds to the frame parameter in `plotly`.
- **Umax**: The maximum displayed value of potential.
- **n, lims, h, kde_fun**: Passed to make_2d_kernel_dist
- **individual_landscape**: Make individual landscape for each simulation?
- **mat_3d**: Also make heatmap matrix?

Value

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

Description

Make 3D kernel smooth distribution

Usage

```r
make_3d_kernel_dist(
  output,
  x,
  y,
  z,
  n = 200,
  lims = c(-0.1, 1.1, -0.1, 1.1, -0.1, 1.1),
  h
)
```

Arguments

- **output**: A matrix of simulation output.
- **x, y, z**: The name of the target variable.
- **n, lims, h**: Passed to `ks::kde()` (but using the format of `MASS::kde2d()` to make it consistent across functions). For `ks::kde`, H = diag(h, 2, 2).

Value

A MASS::kde2d-type list of smooth distribution.
make_3d_matrix

Make a matrix or vector of 3d heatmap graphs for two parameters

Description

(Note: a matrix of interactive maps is currently not supported.)

Usage

make_3d_matrix(
  bs,
  x,
  y,
  rows = NULL,
  cols,
  Umax = 5,
  n = 200,
  lims = c(-0.1, 1.1, -0.1, 1.1),
  h = 0.001,
  kde_fun = "ks",
  individual_landscape = FALSE
)

Arguments

bs              A batch_simulation object created by [batch_simulation].
x, y, rows, cols The names of the target variables. If rows is NULL, only a vector of graphs will be generated.
Umax            The maximum displayed value of potential.
n, lims, h, kde_fun Passed to make_2d_kernel_dist()
individual_landscape Make individual landscape for each simulation?

Value

A 3d_matrix_landscape object that describes the landscape of the system, including the smoothed distribution and the landscape plot.
make_3d_static  
*Make 3D static landscape plots from simulation output*

**Description**
Make 3D static landscape plots from simulation output

**Usage**
```r
make_3d_static(
  output,
  x,
  y,
  Umax = 5,
  n = 200,
  lims = c(-0.1, 1.1, -0.1, 1.1),
  h = 0.001,
  kde_fun = "ks"
)
```

**Arguments**
- `output`: A matrix of simulation output.
- `x, y`: The name of the target variable.
- `Umax`: The maximum displayed value of potential.
- `n, lims, h, kde_fun`: Passed to `make_2d_kernel_dist()`

**Value**
A `3d_static_landscape` object that describes the landscape of the system, including the smooth distribution and the landscape plot.

make_3d_tidy_dist  
*Make a tidy data.frame from smooth 3d distribution matrix*

**Description**
Make a tidy `data.frame` from smooth 3d distribution matrix

**Usage**
```r
make_3d_tidy_dist(dist_3d, value = NULL, var_name = NULL)
```
Arguments

- `dist_3d` : kde2d-type distribution.
- `value` : The value of the variable of interest.
- `var_name` : The name of the variable.

Value

A tidy data.frame.

make_4d_static

Make 4D static space-color plots from simulation output

Description

Make 4D static space-color plots from simulation output

Usage

```r
make_4d_static(
  output,
  x,
  y,
  z,
  Umax = 5,
  n = 50,
  lims = c(-0.1, 1.1, -0.1, 1.1, -0.1, 1.1),
  h = 0.001
)
```

Arguments

- `output` : A matrix of simulation output.
- `x, y, z` : The name of the target variable.
- `Umax` : The maximum displayed value of potential.
- `n, lims, h` : Passed to `make_3d_kernel_dist`

Value

A `4d_static_landscape` object that describes the landscape of the system, including the smoothed distribution and the landscape plot.
make_arg_grid  
*Make variable grids for batch simulation*

**Description**
This is the main function for making the variable grids.

**Usage**
```r
make_arg_grid(arg_set)
```

**Arguments**

**Value**
An `arg_grid` object.

**See Also**
- `batch_simulation()` for a concrete example.

---

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(
  vg,
  start_location_value = 0,
  start_r = 0.1,
  end_location_value = 0.7,
  end_r = 0.15,
  df = NULL,
  print_template = FALSE
)
```
make_barrier_grid_3d

Arguments

vg
A var_grid object.

start_location_value, start_r, end_location_value, end_r
Default values for finding local minimum. See calculate_barrier_3d_batch().

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.

Description

Make a grid for calculating barriers for 3d landscapes

Usage

make_barrier_grid_3d(
  vg,
  start_location_value = c(0, 0),
  start_r = 0.1,
  end_location_value = c(0.7, 0.6),
  end_r = 0.15,
  df = NULL,
  print_template = FALSE
)

Arguments

vg
A var_grid object.

start_location_value, start_r, end_location_value, end_r
Default values for finding local minimum. See calculate_barrier_3d_batch().

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.


**make_var_grid**  
*Make variable grids for batch simulation*

---

**Description**

[Deprecated]  
This function was deprecated. See `new_var_set()`.

**Usage**

```r
make_var_grid(var_set)
```

**Arguments**

- `var_set`  
  A `var_set` object. See `new_var_set()` and `add_var()`.

**Details**

This is the main function for making the variable grids.

**Value**

A `var_grid` object.

**See Also**

- `batch_simulation()` for a concrete example.

---

**modified_simulation**  
*Do the batch simulation*

---

**Description**

This is the main function for the batch simulation.

**Usage**

```r
modified_simulation(sim_fun, ele_list, default_list, bigmemory = TRUE, ...)
batch_simulation(arg_grid, sim_fun, default_list, bigmemory = TRUE, ...)
```
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sim_fun</td>
<td>The simulation function. See <code>sim_fun_test()</code> for an example.</td>
</tr>
<tr>
<td>ele_list</td>
<td>An <code>ele_list</code> object generated by <code>fill_in_struct()</code>.</td>
</tr>
<tr>
<td>default_list</td>
<td>A list of default values for <code>sim_fun</code>.</td>
</tr>
<tr>
<td>bigmemory</td>
<td>Use <code>hash_big.matrix-class()</code> to store large matrices?</td>
</tr>
<tr>
<td>...</td>
<td>Other parameters passed to <code>sim_fun</code></td>
</tr>
<tr>
<td>arg_grid</td>
<td>An <code>arg_grid</code> object. See <code>make_arg_grid()</code>.</td>
</tr>
</tbody>
</table>

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

- `modified_simulation()`: Modify a single 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)
  ),
  length = 1e2,
  seed = 1614,
  bigmemory = FALSE
)
print(batch_output_grad)
```

narg

The number of arguments in an `arg_set`.

Description

The number of arguments in an `arg_set`.

Usage

`narg(arg_set)`
new_arg_set

Arguments
arg_set  An arg_set object.

Value
An integer.

nele  The number of elements in an arg_set.

Description
The number of elements in an arg_set.

Usage
nele(arg_set)

Arguments
arg_set  An arg_set object.

Value
An integer.

new_arg_set  Create and modify argument sets 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 the add to add descriptions of arguments.

Usage
new_arg_set()

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

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.
new_var_set

Value

An arg_set object.

Functions

• new_arg_set(): Create an arg_set.

See Also

make_arg_grid() for making grids from variable sets; batch_simulation() for running batch simulation and a concrete example.

new_var_set Create and modify variable sets for batch simulation

Description

[Deprecated]

This function was deprecated because we decided to shift to a more consistent terminology. Previous par is renamed as arg (argument) and previous var is renamed as ele (element). For creating an arg_set function, please use new_arg_set().

A variable set contains the descriptions of the relevant variables in a batch simulation. Use new_var_set to create a var_set object, and use add_var to add descriptions of variables.

Usage

new_var_set()

add_var(var_set, par_name, var_name, start, end, by)

Arguments

var_set A var_set object.
par_name, var_name

The name of the parameter and variable in the simulation function
start, end, by The data points where you want to test the variables. Passed to seq.

Value

A var_set object.

Functions

• new_var_set(): Create a var_set.
• add_var(): Add a variable to the var_set.
See Also

- `make_var_grid` for making grids from variable sets; `batch_simulation` for running batch simulation and a concrete example.

Examples

```r
test <- new_var_set()
test <- test %>%
  add_var("par1", "var1", 1, 2, 0.1) %>%
  add_var("par2", "var2", 1, 2, 0.1)
```

---

**npar**

*The number of parameters in a var_set.*

---

**Description**

[Deprecated]

This function was deprecated. See `new_var_set()`.

**Usage**

`npar(var_set)`

**Arguments**

- `var_set` A var_set object.

**Value**

An integer.

---

**nvar**

*The number of variables in a var_set.*

---

**Description**

[Deprecated]

This function was deprecated. See `new_var_set()`.

**Usage**

`nvar(var_set)`

**Arguments**

- `var_set` A var_set object.
**plot.barrier**

*Plot the result of a barrier object*

**Value**

An integer.

**Description**

Plot the result of a barrier object

**Usage**

```r
## S3 method for class 'barrier'
plot(x, ...)"```

**Arguments**

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

**Value**

The plot of the local minimums, the saddle point, and the minimum energy path.

---

**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.
print.arg_grid  

*Print an arg_grid object*

## Description

Print an arg_grid object.

## Usage

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

## Arguments

- `x`: The object.
- `detail`: Do you want to print the object details as a full list?
- `...`: Not in use.

## Value

The printed result.

print.arg_set  

*Print an arg_set object.*

## Description

Print an arg_set object.

## Usage

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

## Arguments

- `x`: The object.
- `detail`: Do you want to print the object details as a full list?
- `...`: Not in use.

## Value

The printed result.
print.batch_simulation

Print a batch_simulation object

Description
Print a batch_simulation object

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

Arguments
  x         The object.
  detail    Do you want to print the object details as a full list?
  ...       Not in use.

Value
The printed result.

print.check_conv

Print a check_conv

Description
Print a check_conv

Usage
## S3 method for class 'check_conv'
print(x, ask = TRUE, ...)

Arguments
  x         The object.
  ask       Ask to press enter to see the next plot?
  ...       Not in use.

Value
The printed result.
print.var_grid  
*Print a var_grid object*

**Description**

[Deprecated]
This function was deprecated. See `new_var_set()`.

**Usage**

```r
## S3 method for class 'var_grid'
print(x, detail = FALSE, ...)
```

**Arguments**

- `x` The object.
- `detail` Do you want to print the object details as a full list?
- `...` Not in use.

**Value**

The printed result.

print.var_set  
*Print a var_set object.*

**Description**

[Deprecated]
This function was deprecated. See `new_var_set()`.

**Usage**

```r
## S3 method for class 'var_set'
print(x, detail = FALSE, ...)
```

**Arguments**

- `x` The object.
- `detail` Do you want to print the object details as a full list?
- `...` Not in use.

**Value**

The printed result.
**reverselog_trans**

A function for reversed log transformation

**Usage**

reverselog_trans(base = exp(1))

**Arguments**

- **base**
  The base of logarithm

**Value**

A trans scale object from the scales package.

**save_landscape**

Save landscape plots

**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:

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

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

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

**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 σ₁, σ₂, σ₃ 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

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

`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_test2()` for a more realistic example. `batch_simulation()` for a concrete example.
### Index

<table>
<thead>
<tr>
<th>Function</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_arg_ele(new_arg_set)</td>
<td>24</td>
</tr>
<tr>
<td>add_var(new_var_set)</td>
<td>25</td>
</tr>
<tr>
<td>add_var()</td>
<td>20, 22</td>
</tr>
<tr>
<td>as.hash_big.matrix</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(hash_big.matrix-class)</td>
</tr>
<tr>
<td>attach.hash_big.matrix</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(hash_big.matrix-class)</td>
</tr>
<tr>
<td>attach_all_matrices</td>
<td>3</td>
</tr>
<tr>
<td>batch_simulation</td>
<td>26</td>
</tr>
<tr>
<td>batch_simulation(modified_simulation)</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>bigmemory::as.big.matrix()</td>
<td>3, 11</td>
</tr>
<tr>
<td>calculate_barrier</td>
<td>3</td>
</tr>
<tr>
<td>calculate_barrier_2d</td>
<td>4</td>
</tr>
<tr>
<td>calculate_barrier_2d()</td>
<td>4</td>
</tr>
<tr>
<td>calculate_barrier_2d_batch</td>
<td>5</td>
</tr>
<tr>
<td>calculate_barrier_2d_batch()</td>
<td></td>
</tr>
<tr>
<td>calculate_barrier_3d</td>
<td>6</td>
</tr>
<tr>
<td>calculate_barrier_3d()</td>
<td>4</td>
</tr>
<tr>
<td>calculate_barrier_3d_batch</td>
<td>7</td>
</tr>
<tr>
<td>calculate_barrier_3d_batch()</td>
<td>4, 21</td>
</tr>
<tr>
<td>check_conv</td>
<td>8</td>
</tr>
<tr>
<td>fill_in_struct</td>
<td>8</td>
</tr>
<tr>
<td>fill_in_struct()</td>
<td>23</td>
</tr>
<tr>
<td>find_local_min_2d</td>
<td>9</td>
</tr>
<tr>
<td>find_local_min_3d</td>
<td>9</td>
</tr>
<tr>
<td>get_barrier_height</td>
<td>10</td>
</tr>
<tr>
<td>get_dist</td>
<td>10</td>
</tr>
<tr>
<td>get_geom</td>
<td>11</td>
</tr>
<tr>
<td>ggplot2::ggsave()</td>
<td>31</td>
</tr>
<tr>
<td>hash_big.matrix</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(hash_big.matrix-class)</td>
</tr>
<tr>
<td>hash_big.matrix-class</td>
<td>11</td>
</tr>
<tr>
<td>htmlwidgets::saveWidget()</td>
<td>31</td>
</tr>
<tr>
<td>ks::kde()</td>
<td>13, 16</td>
</tr>
<tr>
<td>make_2d_density</td>
<td>12</td>
</tr>
<tr>
<td>make_2d_kernel_dist</td>
<td>13</td>
</tr>
<tr>
<td>make_2d_kernel_dist()</td>
<td>17, 18</td>
</tr>
<tr>
<td>make_2d_matrix</td>
<td>13</td>
</tr>
<tr>
<td>make_2d_static</td>
<td>14</td>
</tr>
<tr>
<td>make_2d_static()</td>
<td>12</td>
</tr>
<tr>
<td>make_2d_tidy_dist</td>
<td>15</td>
</tr>
<tr>
<td>make_3d_animation</td>
<td>15</td>
</tr>
<tr>
<td>make_3d_kernel_dist</td>
<td>16</td>
</tr>
<tr>
<td>make_3d_kernel_dist()</td>
<td>19</td>
</tr>
<tr>
<td>make_3d_matrix</td>
<td>17</td>
</tr>
<tr>
<td>make_3d_static</td>
<td>18</td>
</tr>
<tr>
<td>make_3d_tidy_dist</td>
<td>18</td>
</tr>
<tr>
<td>make_4d_static</td>
<td>19</td>
</tr>
<tr>
<td>make_arg_grid</td>
<td>20</td>
</tr>
<tr>
<td>make_arg_grid()</td>
<td>23, 25</td>
</tr>
<tr>
<td>make_barrier_grid_2d</td>
<td>20</td>
</tr>
<tr>
<td>make_barrier_grid_3d</td>
<td>21</td>
</tr>
<tr>
<td>make_var_grid</td>
<td>22, 26</td>
</tr>
<tr>
<td>MASS::kde2d()</td>
<td>13, 16</td>
</tr>
<tr>
<td>modified_simulation</td>
<td>22</td>
</tr>
<tr>
<td>modified_simulation()</td>
<td>8</td>
</tr>
<tr>
<td>narg</td>
<td>23</td>
</tr>
<tr>
<td>nele</td>
<td>24</td>
</tr>
<tr>
<td>new_arg_set</td>
<td>24</td>
</tr>
<tr>
<td>new_arg_set()</td>
<td>20, 25</td>
</tr>
<tr>
<td>new_var_set</td>
<td>25</td>
</tr>
<tr>
<td>new_var_set()</td>
<td>22, 26, 30</td>
</tr>
<tr>
<td>npar</td>
<td>26</td>
</tr>
<tr>
<td>nvar</td>
<td>26</td>
</tr>
<tr>
<td>plot.barrier</td>
<td>27</td>
</tr>
<tr>
<td>plot.barrier()</td>
<td>4</td>
</tr>
<tr>
<td>plot.landscape</td>
<td>27</td>
</tr>
<tr>
<td>print.arg_grid</td>
<td>28</td>
</tr>
<tr>
<td>print.arg_set</td>
<td>28</td>
</tr>
<tr>
<td>print.batch_simulation</td>
<td>29</td>
</tr>
</tbody>
</table>
print.check_conv, 29
print.var_grid, 30
print.var_set, 30
reverselog_trans, 31
save_landscape, 31
sim_fun_grad, 32
sim_fun_grad(), 34
sim_fun_nongrad, 33
sim_fun_nongrad(), 32
sim_fun_test, 34
sim_fun_test(), 23
sim_fun_test2(), 34