Package ‘dynwrap’

July 19, 2023

Type Package

Title Representing and Inferring Single-Cell Trajectories

Description Provides functionality to infer trajectories from single-cell data, represent them into a common format, and adapt them. Other biological information can also be added, such as cellular grouping, RNA velocity and annotation. Saelens et al. (2019) <doi:10.1038/s41587-019-0071-9>.

Version 1.2.4

URL `https://github.com/dynverse/dynwrap`

BugReports `https://github.com/dynverse/dynwrap/issues`

License MIT + file LICENSE

LazyData TRUE

RoxygenNote 7.2.3

Encoding UTF-8

Depends R (>= 3.0.0)

Imports assertthat, babelwhale, crayon, dplyr, dynutils (>= 1.0.3), dynparam, igraph, glue, magrittr, Matrix, methods, purrr, processx, readr, stringr, reshape2, tibble, tidyr, yaml

Suggests covr, dyndimred, hdf5r, knitr, pkgload, ranger, rmarkdown, testthat

VignetteBuilder knitr

NeedsCompilation no

Author Robrecht Cannoodt [aut, cre] (<`https://orcid.org/0000-0003-3641-729X`>, rcannood),
Wouter Saelens [aut] (<`https://orcid.org/0000-0002-7114-6248`>, zouter)

Maintainer Robrecht Cannoodt <`rcannood@gmail.com`>

Repository CRAN

Date/Publication 2023-07-19 09:20:02 UTC

R topics documented:

`.method_process_definition` *Method process definition*

Description

Method process definition

Usage

```
.method_process_definition(definition, return_function)
```

Arguments

`definition`	A definition, see `definition()`
`return_function`	Whether to return a function that allows you to override the default parameters, or just return the method meta data as is.

`add_attraction` *Add attraction of cells using RNA velocity*

Description

Add attraction of cells using RNA velocity

Usage

```
add_attraction(dataset)
```

Arguments

`dataset`	A dataset created by `wrap_data()` or `wrap_expression()`

Value

A dynwrap object with the attraction added.

`add_branch_trajectory` *Construct a trajectory given its branch network and the pseudotime of the cells on one of the branches.*

Description

The branch network is converted to a milestone network by giving each branch a start and end milestone. If two branches are connected in the branch network, the end milestone of branch 1 and start milestone of branch 2 will be merged.

Usage

```
add_branch_trajectory(
  dataset,
  branch_network,
  branches,
  branch_progressions,
  ...
)
```

Arguments

`dataset`	A dataset created by `wrap_data()` or `wrap_expression()`
`branch_network`	The network between branches, a dataframe with a *from* and *to* branch identifier
`branches`	The length and directedness of the branches, a dataframe with the branch identifier (*branch_id*), the length of the branch (*length*) and whether it is *directed*
`branch_progressions`	Specifies the progression of a cell along a transition in the branch network. A dataframe containing the *cell_id*, *branch_id* and its progression along the edge (*percentage*, between 0 and 1)
`...`	extra information to be stored in the trajectory

Details

The resulting trajectory will always be directed.

Value

A trajectory object

Examples

```
dataset <- wrap_data(cell_ids = letters)

branch_network <- tibble::tibble(from = c("A", "A"), to = c("B", "C"))
branch_network
branches <- tibble::tibble(branch_id = c("A", "B", "C"), length = 1, directed = TRUE)
branches
branch_progressions <- tibble::tibble(
  cell_id = dataset$cell_ids,
  branch_id = sample(branches$branch_id, length(dataset$cell_ids), replace = TRUE),
  percentage = runif(length(dataset$cell_ids))
)
branch_progressions

trajectory <- add_branch_trajectory(
  dataset,
  branch_network,
  branches,
  branch_progressions
)

# for plotting the result, install dynplot
#- dynplot::plot_graph(trajectory)
```

`add_cell_graph` *Constructs a trajectory using a graph between cells, by mapping cells onto a set of backbone cells.*

Description

The cells that are part of the backbone will form the trajectory. All other cells are moved towards the nearest cell that is part of the backbone.

Usage

```
add_cell_graph(
  dataset,
  cell_graph,
  to_keep,
  milestone_prefix = "milestone_",
  ...
)
```

Arguments

`dataset`	A dataset created by `wrap_data()` or `wrap_expression()`
`cell_graph`	The edges between cells, a dataframe containing the *from* and *to* cells, the *length, and whether this edge is *directed*
`to_keep`	Whether a cells is part of the backbone. May be a character vector with the identifiers of the backbone cells, or a named boolean vector whether a cell is from the backbone
`milestone_prefix`	A prefix to add to the id of the cell ids when they are used as milestones, in order to avoid any naming conflicts,
`...`	extra information to be stored in the wrapper.

Value

A trajectory object

Examples

```
library(dplyr)
dataset <- wrap_data(cell_ids = letters)

backbone_cell_graph <- tibble::tibble(
  from = letters[1:10],
  to = letters[2:11],
  length = 1,
  directed = TRUE
)
```

```
leaves_cell_graph <- tibble::tibble(
  from = letters[12:26],
  to = sample(letters[1:11], 15, replace = TRUE),
  length = 1,
  directed = TRUE
)
cell_graph <- bind_rows(backbone_cell_graph, leaves_cell_graph)
cell_graph
to_keep <- letters[1:11]
to_keep

trajectory <- add_cell_graph(dataset, cell_graph, to_keep)

# for plotting the result, install dynplot
#- dynplot::plot_graph(trajectory)
```

`add_cell_waypoints` *Add or select waypoint cells of a trajectory*

Description

Waypoint cells are cells spread across all of the trajectory such that there is no other cell that has a large geodesic distance to any of the waypoint cells.

Usage

```
add_cell_waypoints(trajectory, num_cells_selected = 100)

is_wrapper_with_waypoint_cells(trajectory)

determine_cell_trajectory_positions(
  milestone_ids,
  milestone_network,
  milestone_percentages,
  progressions,
  divergence_regions
)

select_waypoint_cells(
  milestone_ids,
  milestone_network,
  milestone_percentages,
  progressions,
  divergence_regions,
  num_cells_selected = 100
)
```

Arguments

`trajectory`	The trajectory as created by `infer_trajectory()` or `add_trajectory()`
`num_cells_selected`	About the number of cells selected as waypoints
`milestone_ids`	The ids of the milestones in the trajectory. Type: Character vector.
`milestone_network`	The network of the milestones. Type: Data frame(from = character, to = character, length = numeric, directed = logical).
`milestone_percentages`	A data frame specifying what percentage milestone each cell consists of. Type: Data frame(cell_id = character, milestone_id = character, percentage = numeric).
`progressions`	Specifies the progression of a cell along a transition in the milestone_network. Type: Data frame(cell_id = character, from = character, to = character, percentage = numeric).
`divergence_regions`	A data frame specifying the divergence regions between milestones (e.g. a bifurcation). Type: Data frame(divergence_id = character, milestone_id = character, is_start = logical).

Value

`add_cell_waypoints` returns a trajectory with *waypoint_cells*, a character vector containing the cell ids of the waypoint cells

`select_waypoint_cells` returns a character vector containing the cell ids of the waypoint cells

A dynwrap object with the waypoint cells added.

`add_cluster_graph` *Constructs a trajectory using a cell grouping and a network between groups. Will use an existing grouping if it is present in the dataset.*

Description

A trajectory in this form will rarely be useful, given that cells are only placed at the milestones themselves, but not on the edges between milestones. A better alternative might be to project the cells using a dimensionality reduction, see `add_dimred_projection()`.

Usage

```
add_cluster_graph(
  dataset,
  milestone_network,
  grouping = NULL,
  explicit_splits = FALSE,
  ...
)
```

Arguments

`dataset`	A dataset created by `wrap_data()` or `wrap_expression()`
`milestone_network`	A network of milestones.
`grouping`	A grouping of the cells, can be a named vector or a dataframe with *group_id* and *cell_id*
`explicit_splits`	Whether to make splits specific by adding a starting node. For example: A->B, A->C becomes A->X, X->B, X->C
`...`	extra information to be stored in the wrapper.

Value

A trajectory object

Examples

```
library(tibble)
dataset <- wrap_data(cell_ids = letters)

milestone_network <- tibble::tibble(
  from = c("A", "B", "B"),
  to = c("B", "C", "D"),
  directed = TRUE,
  length = 1
)
milestone_network
grouping <- sample(c("A", "B", "C", "D"), length(dataset$cell_ids), replace = TRUE)
grouping
trajectory <- add_cluster_graph(dataset, milestone_network, grouping)

# for plotting the result, install dynplot
#- dynplot::plot_graph(trajectory)
```

`add_cyclic_trajectory` *Constructs a circular trajectory using the pseudotime values of each cell.*

Description

The pseudotime is divided into three equally sized segments, and are placed within a trajectory in the form A -> B -> C -> A

Usage

```
add_cyclic_trajectory(
  dataset,
  pseudotime,
  directed = FALSE,
  do_scale_minmax = TRUE,
  ...
)
```

Arguments

`dataset`	A dataset created by `wrap_data()` or `wrap_expression()`
`pseudotime`	A named vector of pseudo times.
`directed`	Whether or not the directionality of the pseudotime is predicted.
`do_scale_minmax`	Whether or not to scale the pseudotime between 0 and 1. Otherwise, will assume the values are already within that range.
`...`	extra information to be stored in the wrapper.

Value

A trajectory object

Examples

```
library(tibble)
dataset <- wrap_data(cell_ids = letters)

pseudotime <- tibble(cell_id = dataset$cell_ids, pseudotime = runif(length(dataset$cell_ids)))
pseudotime
trajectory <- add_cyclic_trajectory(dataset, pseudotime)

# for plotting the result, install dynplot
#- dynplot::plot_graph(trajectory)
```

`add_dimred` *Add or create a dimensionality reduction*

Description

This can also perform dimensionality reduction of

- The projected expression state with RNA velocity, only if `dimred` is a function and `pair_with_velocity=TRUE`
- The trajectory, by projecting the milestones and some "waypoints" to the reduced space, only if `dataset` contains a trajectory

Usage

```
add_dimred(
  dataset,
  dimred,
  dimred_milestones = NULL,
  dimred_segment_progressions = NULL,
  dimred_segment_points = NULL,
  project_trajectory = TRUE,
  connect_segments = FALSE,
  pair_with_velocity = !is.null(dataset$expression_future),
  expression_source = "expression",
  ...
)

is_wrapper_with_dimred(dataset)

get_dimred(
  dataset,
  dimred = NULL,
  expression_source = "expression",
  return_other_dimreds = FALSE
)
```

Arguments

`dataset` A dataset created by `wrap_data()` or `wrap_expression()`

`dimred` Can be

- A function which will perform the dimensionality reduction, see `dyndimred::list_dimred_method`
- A matrix with the dimensionality reduction, with cells in rows and dimensions (*comp_1*, *comp_2*, ...) in columns

`dimred_milestones`
An optional dimensionality reduction of the milestones. A matrix with milestones in rows and components (*comp_1*, *comp_2*, ...) in columns

This will be automatically calculated if `project_trajectory` = `TRUE`

`dimred_segment_progressions, dimred_segment_points`
An optional set of points along the trajectory with their dimensionality reduction. `dimred_segment_progressions` is a dataframe containing the *from* and *to* milestones, and their *progression*. `dimred_segment_points` is a matrix with points (the same number as in `dimred_segment_progressions`) in rows and components (*comp_1*, *comp_2*, ...) in columns. Both objects have the same number of rows.

These will be automatically calculated if `project_trajectory` = `TRUE`

`project_trajectory`
Whether to also project the trajectory. Only relevant if dataset contains a trajectory, and dimred_segment_progressions and dimred_segment_points are not provided

`connect_segments`	Whether to connect segments between edges
`pair_with_velocity`	Can perform dimensionality reduction if `dimred` is a function.
`expression_source`	The source of expression, can be "counts", "expression", an expression matrix, or another dataset which contains expression
`...`	extra information to be stored in the wrapper
`return_other_dimreds`	Whether or not to return also the milestone dimreds and the segment dimreds, if available.

Value

A dataset object with *dimred*, which is a numeric matrix with cells in rows and the different components in columns.

- If the dataset contained a trajectory, and `project_trajectory=TRUE` (default), *dimred_milestones*, *dimred_segment_progressions* and *dimred_segment_points* will also be present. These are described in `project_trajectory()`.

See Also

`dyndimred::list_dimred_methods()`, `project_trajectory()`

Examples

```
if (requireNamespace("dyndimred", quietly = TRUE)) {
  dataset <- example_dataset
  dataset <- add_dimred(
    dataset,
    dyndimred::dimred_landmark_mds
  )
  head(dataset$dimred)
}
```

`add_dimred_projection` *Constructs a trajectory by projecting cells within a dimensionality reduction*

Description

A dimensionality reduction of cells and milestones is used, along with the milestone network, to project cells onto the nearest edge. Optionally, a cell grouping can be given which will restrict the edges on which a cell can be projected.

Usage

```
add_dimred_projection(
  dataset,
  milestone_ids = NULL,
  milestone_network,
  dimred,
  dimred_milestones,
  grouping = NULL,
  ...
)
```

Arguments

`dataset`	A dataset created by `wrap_data()` or `wrap_expression()`
`milestone_ids`	The ids of the milestones in the trajectory. Type: Character vector.
`milestone_network`	The network of the milestones. Type: Data frame(from = character, to = character, length = numeric, directed = logical).
`dimred`	Can be • A function which will perform the dimensionality reduction, see `dyndimred::list_dimred_method` • A matrix with the dimensionality reduction, with cells in rows and dimensions (*comp_1*, *comp_2*, ...) in columns
`dimred_milestones`	An optional dimensionality reduction of the milestones. A matrix with milestones in rows and components (*comp_1*, *comp_2*, ...) in columns This will be automatically calculated if `project_trajectory` = `TRUE`
`grouping`	A grouping of the cells, can be a named vector or a dataframe with *group_id* and *cell_id*
`...`	extra information to be stored in the wrapper.

Value

A trajectory object

Examples

```
library(tibble)
dataset <- wrap_data(cell_ids = letters)

milestone_network <- tibble::tibble(
  from = c("A", "B", "B"),
  to = c("B", "C", "D"),
  directed = TRUE,
  length = 1
)
milestone_network
dimred <- matrix(
  runif(length(dataset$cell_ids) * 2),
  ncol = 2,
  dimnames = list(dataset$cell_ids, c("comp_1", "comp_2"))
)
dimred
dimred_milestones <- matrix(
  runif(2*4),
  ncol = 2,
  dimnames = list(c("A", "B", "C", "D"), c("comp_1", "comp_2"))
)
dimred_milestones
trajectory <- add_dimred_projection(
  dataset,
  milestone_network = milestone_network,
  dimred = dimred,
  dimred_milestones = dimred_milestones
)

# for plotting the result, install dynplot
#- dynplot::plot_graph(trajectory)
```

add_end_state_probabilities

Constructs a multifurcating trajectory using end state probabilities

Description

Constructs a multifurcating trajectory using the pseudotime values of each cell and their end state probabilities. If pseudotime values are not given, will use pseudotime already present in the dataset.

Usage

```
add_end_state_probabilities(
  dataset,
  end_state_probabilities,
  pseudotime = NULL,
  do_scale_minmax = TRUE,
  ...
)
```

Arguments

`dataset`	A dataset created by `wrap_data()` or `wrap_expression()`
`end_state_probabilities`	A dataframe containing the *cell_id* and additional numeric columns containing the probability for every end milestone. If the tibble contains only a cell_id column, the data will be processed using `add_linear_trajectory`
`pseudotime`	A named vector of pseudo times.

`do_scale_minmax`	Whether or not to scale the pseudotime between 0 and 1. Otherwise, will assume the values are already within that range.
`...`	Extras to be added to the trajectory

Value

A trajectory object

Examples

```
dataset <- wrap_data(cell_ids = letters)

pseudotime <- runif(length(dataset$cell_ids))
names(pseudotime) <- dataset$cell_ids
pseudotime
end_state_probabilities <- tibble::tibble(
  cell_id = dataset$cell_ids,
  A = runif(length(dataset$cell_ids)),
  B = 1-A
)
end_state_probabilities
trajectory <- add_end_state_probabilities(dataset, end_state_probabilities, pseudotime)

# for plotting the result, install dynplot
#- dynplot::plot_graph(trajectory)
```

`add_expression` *Add count and normalised expression values to a dataset*

Description

Add count and normalised expression values to a dataset

Usage

```
add_expression(
  dataset,
  counts,
  expression,
  feature_info = NULL,
  expression_future = NULL,
  ...
)

is_wrapper_with_expression(dataset)

get_expression(dataset, expression_source = "expression")
```

Arguments

`dataset`	A dataset created by `wrap_data()` or `wrap_expression()`
`counts`	The counts values of genes (columns) within cells (rows). This can be both a dense and sparse matrix.
`expression`	The normalised expression values of genes (columns) within cells (rows). This can be both a dense and sparse matrix.
`feature_info`	Optional meta-information of the features, a dataframe with at least *feature_id* as column
`expression_future`	Projected expression using RNA velocity of genes (columns) within cells (rows). This can be both a dense and sparse matrix.
`...`	extra information to be stored in the dataset
`expression_source`	The source of expression, can be "counts", "expression", an expression matrix, or another dataset which contains expression

Value

A dynwrap object with the expression added.

Examples

```
cell_ids <- c("A", "B", "C")
counts <- matrix(sample(0:10, 3*10, replace = TRUE), nrow = 3)
rownames(counts) <- cell_ids
colnames(counts) <- letters[1:10]
expression <- log2(counts + 1)

dataset <- wrap_data(id = "my_awesome_dataset", cell_ids = cell_ids)
dataset <- add_expression(dataset, counts = counts, expression = expression)

str(dataset$expression)
str(dataset$counts)
```

---

add_feature_importance

Add a feature importance to a dataset

Description

Add a feature importance to a dataset

Usage

```
add_feature_importance(dataset, feature_importance, ...)

is_wrapper_with_feature_importance(dataset)
```

Arguments

`dataset`	A dataset created by `wrap_data()` or `wrap_expression()`
`feature_importance`	The impotances of the features, can be a named vector or a dataframe with columns *feature_id* and *importance*
`...`	Extra information to be stored in the dataset

Value

A dynwrap object with the feature importance added.

Examples

```
dataset <- example_dataset

feature_importance <- runif(nrow(dataset$feature_info))
names(feature_importance) <- dataset$feature_info$feature_id

dataset <- add_feature_importance(dataset, feature_importance)
head(dataset$feature_importance)
```

`add_grouping` *Add a cell grouping to a dataset*

Description

Add a cell grouping to a dataset

Usage

```
add_grouping(dataset, grouping, group_ids = NULL, ...)

is_wrapper_with_grouping(dataset)

get_grouping(dataset, grouping = NULL)
```

Arguments

`dataset`	A dataset created by `wrap_data()` or `wrap_expression()`
`grouping`	A grouping of the cells, can be a named vector or a dataframe with *group_id* and *cell_id*
`group_ids`	All group identifiers, optional
`...`	Extra information to be stored in the dataset

Value

A dynwrap object with the grouping added.

Examples

```
dataset <- example_dataset

grouping <- sample(c("A", "B", "C"), length(dataset$cell_ids), replace = TRUE)
names(grouping) <- dataset$cell_ids

dataset <- add_grouping(dataset, grouping)
head(dataset$grouping)
```

`add_linear_trajectory` *Constructs a linear trajectory using pseudotime values*

Description

Constructs a linear trajectory using pseudotime values

Usage

```
add_linear_trajectory(
  dataset,
  pseudotime,
  directed = FALSE,
  do_scale_minmax = TRUE,
  ...
)
```

Arguments

`dataset`	A dataset created by `wrap_data()` or `wrap_expression()`
`pseudotime`	A named vector of pseudo times.
`directed`	Whether the trajectory will be directed.
`do_scale_minmax`	Whether or not to scale the pseudotime between 0 and 1. Otherwise, will assume the values are already within that range.
`...`	extra information to be stored in the trajectory

Value

A trajectory object

Examples

```
library(tibble)
dataset <- wrap_data(cell_ids = letters)

pseudotime <- tibble(
  cell_id = dataset$cell_ids,
  pseudotime = runif(length(dataset$cell_ids))
)

trajectory <- add_linear_trajectory(dataset, pseudotime)
```

---

add_prior_information *Add or compute prior information for a trajectory*

Description

If you specify

For example, what are the start cells, the end cells, to which milestone does each cell belong to, ...

Usage

```
add_prior_information(
  dataset,
  start_id = NULL,
  end_id = NULL,
  groups_id = NULL,
  groups_network = NULL,
  features_id = NULL,
  groups_n = NULL,
  start_n = NULL,
  end_n = NULL,
  leaves_n = NULL,
  timecourse_continuous = NULL,
  timecourse_discrete = NULL,
  dimred = NULL,
  verbose = TRUE
)

is_wrapper_with_prior_information(dataset)

generate_prior_information(
  cell_ids,
  milestone_ids,
  milestone_network,
  milestone_percentages,
  progressions,
  divergence_regions,
  expression,
  feature_info = NULL,
  cell_info = NULL,
  marker_fdr = 0.005,
  given = NULL,
  verbose = FALSE
)
```

Arguments

`dataset`	A dataset created by `wrap_data()` or `wrap_expression()`
`start_id`	The start cells
`end_id`	The end cells
`groups_id`	The grouping of cells, a dataframe with cell_id and group_id
`groups_network`	The network between groups, a dataframe with from and to
`features_id`	The features (genes) important for the trajectory
`groups_n`	Number of branches
`start_n`	Number of start states
`end_n`	Number of end states
`leaves_n`	Number of leaves
`timecourse_continuous`	The time for every cell
`timecourse_discrete`	The time for every cell in groups
`dimred`	A dimensionality reduction of the cells (see `add_dimred()`)
`verbose`	Whether or not to print informative messages
`cell_ids`	The identifiers of the cells.
`milestone_ids`	The ids of the milestones in the trajectory. Type: Character vector.
`milestone_network`	The network of the milestones. Type: Data frame(from = character, to = character, length = numeric, directed = logical).
`milestone_percentages`	A data frame specifying what percentage milestone each cell consists of. Type: Data frame(cell_id = character, milestone_id = character, percentage = numeric).
`progressions`	Specifies the progression of a cell along a transition in the milestone_network. Type: Data frame(cell_id = character, from = character, to = character, percentage = numeric).
`divergence_regions`	A data frame specifying the divergence regions between milestones (e.g. a bifurcation). Type: Data frame(divergence_id = character, milestone_id = character, is_start = logical).
`expression`	The normalised expression values of genes (columns) within cells (rows). This can be both a dense and sparse matrix.

`feature_info`	Optional meta-information pertaining the features.
`cell_info`	Optional meta-information pertaining the cells.
`marker_fdr`	Maximal FDR value for a gene to be considered a marker
`given`	Prior information already calculated

Details

If the dataset contains a trajectory (see `add_trajectory()`) and expression data, this function will compute and add prior information using `generate_prior_information()`

The dataset has to contain a trajectory for this to work

Value

A dynwrap object with the prior information added.

Examples

```
# add some prior information manually
dataset <- example_dataset
dataset <- add_prior_information(dataset, start_id = "Cell1")
dataset$prior_information$start_id

# compute prior information from a trajectory
trajectory <- example_trajectory
trajectory <- add_prior_information(trajectory)
trajectory$prior_information$end_id
```

add_regulatory_network *Add a GRN to a dynwrap object*

Description

Add a GRN to a dynwrap object

Usage

```
add_regulatory_network(
  dataset,
  regulatory_network,
  regulatory_network_sc = NULL,
  regulators = NULL,
  targets = NULL,
  ...
)
```

Arguments

`dataset`	A dataset created by `wrap_data()` or `wrap_expression()`
`regulatory_network`	A data frame consisting of three columns: `"regulator"`, `"target"`, `"strength"`.
`regulatory_network_sc`	A data frame consisting of four columns: `"cell_id"`, `"regulator"`, `"target"`, `"strength"`.
`regulators`	The feature ids of the regulators.
`targets`	The feature ids of the targets.
`...`	Extra arguments to be saved in the model.

Value

A dynwrap object with the regulatory network added.

`add_root` *Root the trajectory*

Description

Designates a milestone as root, and changes the direction of any edges so that they move away from the specified root (if `flip_edges=TRUE`, default).

Usage

```
add_root(
  trajectory,
  root_cell_id = trajectory$root_cell_id,
  root_milestone_id = trajectory$root_milestone_id,
  flip_edges = TRUE
)

add_root_using_expression(
  trajectory,
  features_oi,
  expression_source = "expression"
)

is_rooted(trajectory)

remove_root(trajectory)
```

Arguments

`trajectory`	The trajectory as created by `infer_trajectory()` or `add_trajectory()`
`root_cell_id`	The root cell id, not required if root_milestone_id is given
`root_milestone_id`	The root milestone id, not required if root_cell_id is given
`flip_edges`	Whether to flip edges which are going in the other direction compared to the root
`features_oi`	The feature ids which will be used to root
`expression_source`	Source of the expression, either a string or a matrix

Details

A `root_cell_id` can also be specified, and the root milestone will be determined as the milestone with the closest geodesic distance to this cell.

Value

A trajectory, with a *root_milestone_id* and with adapted *milestone_network* and *progressions* based on the rooting.

Examples

```
# add a root using a root cell
trajectory <- example_trajectory
trajectory <- add_root(
  trajectory,
  root_cell_id = sample(trajectory$cell_ids, 1)
)
trajectory$root_milestone_id

# add a root using a root milestone id
trajectory <- add_root(
  trajectory,
  root_milestone_id = "milestone_end"
)
trajectory$root_milestone_id
trajectory$milestone_network
```

`add_tde_overall` *Add information on overall differentially expressed features*

Description

To calculate differential expression within trajectories, check out the dynfeature package.

Usage

```
add_tde_overall(trajectory, tde_overall)
```

Arguments

`trajectory`	The trajectory as created by `infer_trajectory()` or `add_trajectory()`
`tde_overall`	A dataframe containing the *feature_id*, and some other columns including whether it is differentially expressed (*differentially_expressed*), the rank of differential expression among all other features (*rank*), the p-value (*pval*) or corrected value (*qval*), and the log-fold change (*lfc*).

Value

A trajectory containing *tde_overall*, a dataframe containing the *feature_id*, and some other columns including whether it is differentially expressed (*differentially_expressed*), the rank of differential expression among all other features (*rank*), the p-value (*pval*) or corrected value (*qval*), and the log-fold change (*lfc*).

Examples

```
trajectory <- example_trajectory
tde_overall <- tibble::tibble(
  feature_id = trajectory$feature_info$feature_id,
  differentially_expressed = sample(c(TRUE, FALSE), length(feature_id), replace = TRUE)
)
trajectory <- add_tde_overall(trajectory, tde_overall)
trajectory$tde_overall
```

`add_timings` *Add timings to a trajectory*

Description

Add timings to a trajectory

Helper function for storing timings information.

Usage

```
add_timings(trajectory, timings)

is_wrapper_with_timings(trajectory)

add_timing_checkpoint(timings, name)
```

Arguments

`trajectory`	The trajectory as created by `infer_trajectory()` or `add_trajectory()`
`timings`	A list of timings.
`name`	The name of the timings checkpoint.

Value

A dynwrap object with the timings added.

Examples

```
trajectory <- example_trajectory
trajectory <- add_timings(
  trajectory,
  list(start = 0, end = 1)
)
```

`add_trajectory` *Construct a trajectory given its milestone network and milestone percentages or progressions*

Description

Construct a trajectory given its milestone network and milestone percentages or progressions

Usage

```
add_trajectory(
  dataset,
  milestone_ids = NULL,
  milestone_network,
  divergence_regions = NULL,
  milestone_percentages = NULL,
  progressions = NULL,
  allow_self_loops = FALSE,
  ...
)

is_wrapper_with_trajectory(trajectory)
```

Arguments

`dataset`	A dataset created by `wrap_data()` or `wrap_expression()`
`milestone_ids`	The ids of the milestones in the trajectory. Type: Character vector.

`milestone_network`	The network of the milestones. Type: Data frame(from = character, to = character, length = numeric, directed = logical).
`divergence_regions`	A data frame specifying the divergence regions between milestones (e.g. a bifurcation). Type: Data frame(divergence_id = character, milestone_id = character, is_start = logical).
`milestone_percentages`	A data frame specifying what percentage milestone each cell consists of. Type: Data frame(cell_id = character, milestone_id = character, percentage = numeric).
`progressions`	Specifies the progression of a cell along a transition in the milestone_network. Type: Data frame(cell_id = character, from = character, to = character, percentage = numeric).
`allow_self_loops`	Whether to allow self loops Type: Logical
`...`	extra information to be stored in the dataset
`trajectory`	The trajectory as created by `infer_trajectory()` or `add_trajectory()`

Value

The dataset object with trajectory information, including:

- *milestone_ids*: The names of the milestones, a character vector.
- *milestone_network*: The network between the milestones, a dataframe with the *from* milestone, *to* milestone, *length* of the edge, and whether it is *directed*.
- *divergence_regions*: The regions between three or more milestones where cells are diverging, a dataframe with the divergence id (*divergence_id*), the milestone id (*milestone_id*) and whether this milestone is the start of the divergence (*is_start*)
- *milestone_percentages*: For each cell its closeness to a particular milestone, a dataframe with the cell id (*cell_id*), the milestone id (*milestone_id*), and its *percentage* (a number between 0 and 1 where higher values indicate that a cell is close to the milestone).
- *progressions*: For each cell its progression along a particular edge of the *milestone_network*. Contains the same information as *milestone_percentages*. A dataframe with cell id (*cell_id*), *from* milestone, *to* milestone, and its *percentage* (a number between 0 and 1 where higher values indicate that a cell is close to the *to* milestone and far from the *from* milestone).

Examples

```
library(dplyr)
library(tibble)

dataset <- wrap_data(cell_ids = letters)

milestone_network <- tribble(
  ~from, ~to, ~length, ~directed,
  "A", "B", 1, FALSE,
  "B", "C", 2, FALSE,
  "B", "D", 1, FALSE,
)
milestone_network
progressions <- milestone_network %>%
  sample_n(length(dataset$cell_ids), replace = TRUE, weight = length) %>%
  mutate(
    cell_id = dataset$cell_ids,
    percentage = runif(n())
  ) %>%
  select(cell_id, from, to, percentage)
progressions
divergence_regions <- tribble(
  ~divergence_id, ~milestone_id, ~is_start,
  "1", "A", TRUE,
  "1", "B", FALSE,
  "1", "C", FALSE
)
divergence_regions

trajectory <- add_trajectory(
  dataset,
  milestone_network = milestone_network,
  divergence_regions = divergence_regions,
  progressions = progressions
)

# for plotting the result, install dynplot
#- dynplot::plot_graph(trajectory)
```

allowed_inputs — *All allowed inputs for a TI method*

Description

All allowed inputs for a TI method

Usage

```
allowed_inputs
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 16 rows and 2 columns.

Examples

```
allowed_inputs
```

`allowed_outputs` *All allowed outputs for a TI method*

Description

All allowed outputs for a TI method

Usage

```
allowed_outputs
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 14 rows and 5 columns.

Examples

```
allowed_outputs
```

`calculate_attraction` *Calculate the attraction of cells to other cells using velocity*

Description

Calculate the attraction of cells to other cells using velocity

Usage

```
calculate_attraction(
  current,
  projected,
  cells = colnames(projected),
  n_waypoints = 50,
  k = 50
)
```

Arguments

`current`	Current expression
`projected`	Projected expression based on RNA velocity
`cells`	Which cells to use
`n_waypoints`	Number of waypoints to use
`k`	K knns

Value

Matrix containing the attraction ([-1, 1]) of each cell to the waypoint cells

---

`calculate_average_by_group` *Calculate average values of a matrix*

Description

`calculate_average_by_group` will calculate an average value per group, given a matrix with cells in the rows and some features in the columns (e.g. expression matrix)

Usage

```
calculate_average_by_group(x, cell_grouping)
```

Arguments

`x`	A matrix. One row for every cell; one column for every feature. The rows must be named.
`cell_grouping`	A data frame denoting the grouping of the cells. Format: `tibble(cell_id = character(), group_id = character())`.

Value

A matrix containing for each feature (column) the average

Examples

```
calculate_average_by_group(
  x = example_trajectory$expression,
  cell_grouping = example_trajectory$prior_information$groups_id
)
```

`calculate_geodesic_distances` *Calculate geodesic distances between cells in a trajectory*

Description

Will calculate geodesic distances between cells within a trajectory. To speed things up, only the distances with a set of waypoint cells are calculated.

Usage

```
calculate_geodesic_distances(
  trajectory,
  waypoint_cells = NULL,
  waypoint_milestone_percentages = NULL,
  directed = FALSE
)

compute_tented_geodesic_distances(
  trajectory,
  waypoint_cells = NULL,
  waypoint_milestone_percentages = NULL
)
```

Arguments

`trajectory`	The trajectory as created by `infer_trajectory()` or `add_trajectory()`
`waypoint_cells`	A vector of waypoint cells. Only the geodesic distances between waypoint cells and all other cells will be calculated.
`waypoint_milestone_percentages`	The milestone percentages of non-cell waypoints, containing waypoint_id, milestone_id and percentage columns
`directed`	Take into account the directions of the milestone edges. The cells that cannot be reached from a particular waypoint will have distance infinity. You can also give a character, if it is "forward" it will look forward, if it is "reverse" it will look in the reversed direction

Details

The geodesic distance takes into account the length of an edge regions of delayed commitment.

Value

A matrix containing geodesic distances between each waypoint cell (rows) and cell (columns)

Examples

```
geodesic_distances <- calculate_geodesic_distances(example_trajectory)
geodesic_distances[1:10, 1:10]
```

`calculate_pseudotime` *Add or calculate pseudotime as distance from the root*

Description

When calculating the pseudotime, the trajectory is expected to be rooted (see `add_root()`)

Usage

```
calculate_pseudotime(trajectory)

add_pseudotime(trajectory, pseudotime = NULL)
```

Arguments

`trajectory`	The trajectory as created by `infer_trajectory()` or `add_trajectory()`
`pseudotime`	Named vector containing the pseudotime for every cell. If not given, the pseudotime will be calculated.

Value

The trajectory with *pseudotime* added, which is a named vector containing the pseudotime values for every cell.

See Also

`add_root()`, `add_linear_trajectory()`

`calculate_trajectory_dimred` *Layout the trajectory and its cells in 2 dimensions using a graph layout*

Description

Layout the trajectory and its cells in 2 dimensions using a graph layout

Usage

```
calculate_trajectory_dimred(trajectory, adjust_weights = FALSE)
```

Arguments

`trajectory`	The trajectory as created by `infer_trajectory()` or `add_trajectory()`
`adjust_weights`	Whether or not to rescale the milestone network weights

Value

A list containg

- *milestone_positions*: A dataframe containing the *milestone_id* and the location of each milestone (*comp_1* and *comp_2*)
- *edge_positions*: A dataframe containing for each edge (*from*, *to*, *length* and *directed* columns) the position of the from milestone (*comp_1_from* and *comp_2_from*) and to milestone (*comp_1_to* and *comp_2_to*).
- *cell_positions*: A dataframe containing the *cell_id* and the ¨location of each cell (*comp_1* and *comp_2*)
- *divergence_edge_positions*: A dataframe as *edge_positions* but for each edge within a divergence
- *divergence_polygon_positions*: A dataframe containing the *triangle_id* and the location of the milestone within a divergence (*comp_1* and *comp_2*)

See Also

`wrap_data()`

Examples

```
trajectory_dimred <- calculate_trajectory_dimred(example_trajectory)
head(trajectory_dimred$milestone_positions)
head(trajectory_dimred$edge_positions)
head(trajectory_dimred$cell_positions)
```

---

classify_milestone_network

Classify a milestone network

Description

Classify a milestone network

Usage

```
classify_milestone_network(milestone_network)
```

Arguments

`milestone_network`	A milestone network

Value

A list containing

- *network_type*: The network type (also known as the trajectory_type). See `dynwrap::trajectory_types` for an overview.
- *directed*: Whether the trajectory is directed
- *properties*: Different properties of the trajectory, including:
  - *is_directed*: Whether the trajectory is directed
  - *max_degree*: The maximal degree
  - *num_branch_nodes*: The number of branching nodes
  - *num_outer_nodes*: Number of leaf (outer) nodes
  - *is_self_loop*: Whether it contains self-loops
  - *has_cycles*: Whether it has cycles
  - *num_components*: The number of independent components

See Also

`dynwrap::trajectory_types`

Examples

```
milestone_network <- tibble::tibble(
  from = c("A", "B", "C"),
  to = c("B", "C", "A"),
  length = 1,
  directed = TRUE
)
classification <- classify_milestone_network(milestone_network)
classification$network_type
classification$directed

milestone_network <- tibble::tibble(
  from = c("A", "B", "B", "C", "C"),
  to = c("B", "C", "D", "E", "F"),
  length = 2,
  directed = FALSE
)
classification <- classify_milestone_network(milestone_network)
classification$network_type
classification$directed
classification$props
```

`convert_definition` *Convert a definition loaded in from a yaml*

Description

Convert a definition loaded in from a yaml

Usage

```
convert_definition(definition_raw)
```

Arguments

`definition_raw`	The raw definition loaded from the yaml

`convert_milestone_percentages_to_progressions` *Conversion between milestone percentages and progressions*

Description

Conversion between milestone percentages and progressions

Usage

```
convert_milestone_percentages_to_progressions(
  cell_ids,
  milestone_ids,
  milestone_network,
  milestone_percentages
)
```

Arguments

`cell_ids`	The identifiers of the cells.
`milestone_ids`	The ids of the milestones in the trajectory. Type: Character vector.
`milestone_network`	The network of the milestones. Type: Data frame(from = character, to = character, length = numeric, directed = logical).
`milestone_percentages`	A data frame specifying what percentage milestone each cell consists of. Type: Data frame(cell_id = character, milestone_id = character, percentage = numeric).

Value

A data frame with columns `cell_id`, `from`, `to`, `percentage`.

See Also

`add_trajectory()`, convert_progressions_to_milestone_percentages

Examples

```
progressions <- convert_milestone_percentages_to_progressions(
  cell_ids = example_trajectory$cell_ids,
  milestone_ids = example_trajectory$milestone_ids,
  milestone_network = example_trajectory$milestone_network,
  milestone_percentages = example_trajectory$milestone_percentages
)
head(progressions)
```

---

convert_progressions_to_milestone_percentages

Conversion between milestone percentages and progressions

Description

Conversion between milestone percentages and progressions

Usage

```
convert_progressions_to_milestone_percentages(
  cell_ids,
  milestone_ids,
  milestone_network,
  progressions
)
```

Arguments

`cell_ids`	The identifiers of the cells.
`milestone_ids`	The ids of the milestones in the trajectory. Type: Character vector.
`milestone_network`	The network of the milestones. Type: Data frame(from = character, to = character, length = numeric, directed = logical).
`progressions`	Specifies the progression of a cell along a transition in the milestone_network. Type: Data frame(cell_id = character, from = character, to = character, percentage = numeric).

Value

A data frame with columns `cell_id`, `milestone_id`, and `percentage`.

See Also

`add_trajectory()`, `convert_milestone_percentages_to_progressions()`

Examples

```
milestone_percentages <- convert_progressions_to_milestone_percentages(
  cell_ids = example_trajectory$cell_ids,
  milestone_ids = example_trajectory$milestone_ids,
  milestone_network = example_trajectory$milestone_network,
  progressions = example_trajectory$progressions
)
head(milestone_percentages)
```

---

create_ti_method_container

Create a TI method from a docker / singularity container

Description

These functions create a TI method from a container using `babelwhale`. Supports both docker and singularity as a backend. See `vignette("create_ti_method_container", "dynwrap")` for a tutorial on how to create a containerized TI method.

Usage

```
create_ti_method_container(
  container_id,
  pull_if_needed = TRUE,
  return_function = TRUE
)
```

Arguments

`container_id`	The name of the container repository (e.g. `"dynverse/ti_angle"`).
`pull_if_needed`	Pull the container if not yet available.
`return_function`	Whether to return a function that allows you to override the default parameters, or just return the method meta data as is.

Value

A function that can be used to adapt the parameters of the method. This functions returns a list containing all metadata of the method, and can be used to infer a trajectory

See Also

vignette("create_ti_method_container", "dynwrap")

Examples

```
library(babelwhale)

# only run if docker works on this platform
if (test_docker_installation()) {
  method <- create_ti_method_container("dynverse/ti_angle")
  trajectory <- infer_trajectory(example_dataset, method())
}
```

---

create_ti_method_definition

Create a TI method from a local method definition file

Description

The local method definition file describes a method that is runnable on the local system. See `vignette("create_ti_method_definition", "dynwrap")` for a tutorial on how to create a containerized TI method.

Usage

```
create_ti_method_definition(definition, script, return_function = TRUE)
```

Arguments

`definition`	A definition, see `definition()`
`script`	Location of the script that will be executed. Has to contain a #!
`return_function`	Whether to return a function that allows you to override the default parameters, or just return the method meta data as is.

Value

A method definition

Examples

```
# See the vignette "create_ti_method_definition" to get a good idea on how
# to use this function.

# create a definition.yaml file and a run.R/py script.
# method <- create_ti_method_definition("definition.yml", "run.R")
# trajectory <- infer_trajectory(example_dataset, method(), verbose = TRUE)
```

create_ti_method_r — *Create a TI method from an R function wrapper*

Description

Create a TI method from an R function wrapper

Usage

```
create_ti_method_r(
  definition,
  run_fun,
  package_required = character(),
  package_loaded = character(),
  remotes_package = character(),
  return_function = TRUE
)
```

Arguments

`definition`	A definition, see `definition()`
`run_fun`	A function to infer a trajectory, with parameters counts/expression, parameters, priors, verbose and seed
`package_required`	The packages that need to be installed before executing the method.
`package_loaded`	The packages that need to be loaded before executing the method.
`remotes_package`	Package from which the remote locations of dependencies have to be extracted, eg. `dynmethods`.
`return_function`	Whether to return a function that allows you to override the default parameters, or just return the method meta data as is.

Value

A method definition

Examples

```
# define the parameters and other metadata
definition <- definition(
  method = def_method(
    id = "comp1"
  ),
  parameters = def_parameters(
    dynparam::integer_parameter(
      id = "component",
      default = 1,
      distribution = dynparam::uniform_distribution(1, 10),
      description = "The nth component to use"
    )
  ),
  wrapper = def_wrapper(
    input_required = "expression",
    input_optional = "start_id"
  )
)

# define a wrapper function
run_fun <- function(expression, priors, parameters, seed, verbose) {
  pca <- prcomp(expression)

  pseudotime <- pca$x[, parameters$component]

  # flip pseudotimes using start_id
  if (!is.null(priors$start_id)) {
    if(mean(pseudotime[start_id]) > 0.5) {
     pseudotime <- 1-pseudotime
    }
  }

  wrap_data(cell_ids = rownames(expression)) %>%
    add_linear_trajectory(pseudotime = pseudotime)
}

method <- create_ti_method_r(definition, run_fun, package_loaded = "dplyr")
trajectory <- infer_trajectory(example_dataset, method())
```

`definition` *Create a definition*

Description

A definition contains meta information on a TI method and various aspects thereof. For brevity, the example only contains a minimum example, check the documentation of the `def_*` helper functions for more extensive examples.

Usage

```
definition(
  method,
  wrapper,
  manuscript = NULL,
  container = NULL,
  package = NULL,
  parameters = parameter_set()
)

is_ti_method(method)
```

Arguments

method	Meta information on the TI method (see def_method()).
wrapper	Meta information on the wrapper itself (see def_wrapper()).
manuscript	Meta information on the manuscript, if applicable (see def_manuscript()).
container	Meta information on the container in which the wrapper resides, if applicable (see def_container()).
package	Meta information on the package in which the wrapper resides, if applicable (see def_package()).
parameters	Meta information on the parameters of the TI method (see def_parameters()).

Value

A method definition

Examples

```
library(dynparam)
definition(
  method = def_method(id = "some_method"),
  wrapper = def_wrapper(input_required = "expression"),
  parameters = parameter_set(
   integer_parameter(id = "k", default = 5L, distribution = uniform_distribution(3L, 20L))
  )
)
```

---

def_author *Meta information on an author*

Description

Meta information on an author

Usage

```
def_author(given, family, email = NULL, github = NULL, orcid = NULL)
```

Arguments

given	The given name
family	The family name
email	The email address
github	The github handle
orcid	The orcid id

Examples

```
def_author(
  given = "Bob",
  family = "Dylan",
  email = "bob@dylan.com",
  github = "bobdylan",
  orcid = "0000-0003-1234-5678"
)
```

`def_container` *Meta information on the container in which the wrapper resides*

Description

Meta information on the container in which the wrapper resides

Usage

```
def_container(docker, url = NULL)
```

Arguments

`docker`	The handle of the docker container
`url`	An url of where the docker codebase resides (containing definition.yml, Dockerfile, ...)

Examples

```
def_container(
  docker = "bobdylan/ti_some_method",
  url = "https://github.com/bobdylan/ti_some_method"
)
```

`def_manuscript` *Meta information on the manuscript*

Description

Meta information on the manuscript

Usage

```
def_manuscript(
  doi = NULL,
  google_scholar_cluster_id = NULL,
  preprint_date = NULL,
  publication_date = NULL
)
```

Arguments

`doi`	A doi identifier (not an url)
`google_scholar_cluster_id`	The google cluster id. Finding this id is a bit tricky; you need to find the manuscript on one of the author pages, and hover over the 'All X versions' button. Example: google scholar page, screenshot.
`preprint_date`	Date of publication of the preprint (format: YYYY-MM-DD).
`publication_date`	Date of publication of the peer-reviewed manuscript (format: YYYY-MM-DD).

Examples

```
def_manuscript(
  doi = "101010101/1101010101",
  google_scholar_cluster_id = "1010001010101111211",
  preprint_date = "1970-01-30",
  publication_date = "1970-01-31"
)
```

def_method *Define meta information on the TI method.*

Description

Define meta information on the TI method.

Usage

```
def_method(
  id,
  name = id,
  source = "tool",
  tool_id = NULL,
  platform = NULL,
  url = NULL,
  license = NULL,
  authors = list(),
  description = NULL
)
```

Arguments

`id`	An id by which to identify a method. Should only contain lowercase letters or underscores.
`name`	The name of the method.
`source`	The type of TI method. Options are : • `"tool"`: a published TI method (peer-reviewed or preprint) (default), • `"adaptation"`: an adaptation of a published method, • `"offtheshelf"`: a method constructed from off-the-shelf algorithms, • `"control"`: a control TI method (so not actually a TI method).
`tool_id`	If there are multiple TI methods from the same toolkit, the name of the toolkit can be specified here.
`platform`	The platform the TI method uses (e.g. R, Python, C++, ...).
`url`	An URL to the codebase of the method.
`license`	The software license the method uses (e.g. GPL-3, BSD-3, Artistic-2.0, MIT).
`authors`	A list of authors (see example).
`description`	Additional information on the method

Examples

```
def_method(
  id = "some_method",
  name = "Some method <3",
  source = "tool",
  tool_id = "bobstoolkit",
  platform = "VBA",
  url = "https://github.com/bobdylan/singlecellvba",
  license = "GPL-3",
  authors = list(
    def_author(
      given = "Bob",
      family = "Dylan",
      email = "bob@dylan.com",
      github = "bobdylan",
      orcid = "0000-0003-1234-5678"
    )
  ),
  description = "I love trajectories!!"
)
```

`def_package` *Meta information on the package in which the TI function resides*

Description

Meta information on the package in which the TI function resides

Usage

```
def_package(remote, name, function_name)
```

Arguments

`remote`	The github repository handle
`name`	The name of the package
`function_name`	The name of the function

Examples

```
def_package(
  remote = "rcannood/SCORPIUS",
  name = "SCORPIUS",
  function_name = "ti_scorpius"
)
```

`def_parameters` *Meta information on the parameters of the TI method*

Description

Parameters can be defined using `dynparam::dynparam()`.

Usage

```
def_parameters(..., parameters = NULL, forbidden = NULL)
```

Arguments

`...`	Parameters to wrap in a parameter set.
`parameters`	A list of parameters to wrap in a parameter set.
`forbidden`	States forbidden region of parameter via a character vector, which will be turned into an expression.

Examples

```
library(dynparam)
def_parameters(
  character_parameter(id = "method", default = "one", values = c("one", "two", "three")),
  integer_parameter(
    id = "ndim",
    default = 3L,
    distribution = uniform_distribution(lower = 2L, upper = 20L)
  ),
  numeric_parameter(
    id = "beta",
    default = 0.005,
    distribution = expuniform_distribution(lower = 1e-10, upper = 1)
  )
)
```

def_wrapper *Meta information on the wrapper*

Description

Meta information on the wrapper

Usage

```
def_wrapper(
  input_required,
  input_optional = character(),
  type = "trajectory",
  topology_inference = NULL,
  trajectory_types = character()
)
```

Arguments

`input_required`	The required inputs for this method. See `dynwrap::allowed_inputs()`.
`input_optional`	Optional inputs for this method. See `dynwrap::allowed_inputs()`.
`type`	Which type of trajectory post-processing is used. Possible values: `"trajectory"` (default), `"linear_trajectory"`, `"cyclic_trajectory"`, `"branch_trajectory"`, `"cluster_graph"`, `"dimred_projection"`, `"end_state_probabilities"`, `"cell_graph"`.
`topology_inference`	Whether the topology is fixed (`"fixed"`), free (`"free"`), or fixed by a parameter provided to the algorithm (`"param"`).
`trajectory_types`	The possible trajectory types this method can return. Must be a subset of `c("cyclic", "linear", "bifurcation", "convergence", "multifurcation", "tree", "graph", "acyclic_graph", "disconnected_graph")`

Examples

```
def_wrapper(
  input_required = c("expression", "start_id"),
  input_optional = "groups_n",
  type = "dimred_projection",
  trajectory_types = c("linear", "cyclic"),
  topology_inference = "free"
)
```

`dynwrap` *Inferring and adapting single-cell trajectories*

Description

`example_dataset` *Example dataset*

Description

Example dataset

Usage

```
example_dataset
```

Format

An object of class `dynwrap::with_dimred` (inherits from `dynwrap::with_expression`, `dynwrap::data_wrapper`, `list`) of length 11.

`example_trajectory` *Example trajectory*

Description

Example trajectory

Usage

```
example_trajectory
```

Format

An object of class `dynwrap::with_dimred` (inherits from `dynwrap::with_cell_waypoints`, `dynwrap::with_prior`, `dynwrap::with_trajectory`, `dynwrap::with_dimred`, `dynwrap::with_expression`, `dynwrap::data_wrapper`, `list`) of length 21.

`flip_edges` *Flip a set of edges of the milestone network*

Description

Note that this will remove associated roots, reroot the trajectory using `add_root()`

Usage

```
flip_edges(trajectory, milestone_network_toflip)
```

Arguments

`trajectory`	The trajectory as created by `infer_trajectory()` or `add_trajectory()`
`milestone_network_toflip`	A dataframe with a from and to column, containing the subset of the milestone network #'

Value

A trajectory object

`gather_cells_at_milestones` *Gather cells to their closest milestones*

Description

Cells will be moved to their closest milestones.

Usage

```
gather_cells_at_milestones(trajectory)
```

Arguments

`trajectory`	The trajectory as created by `infer_trajectory()` or `add_trajectory()`

Value

A trajectory where cells where moved to the closest milestone, the milestone_percentages and progressions will be adapated.

Examples

```
trajectory <- example_trajectory
trajectory <- gather_cells_at_milestones(trajectory)
head(trajectory$milestone_percentages)
```

`generate_parameter_documentation` *Generate the parameter documentation of a method, use with* `@eval`

Description

Generate the parameter documentation of a method, use with `@eval`

Usage

```
generate_parameter_documentation(definition)
```

Arguments

`definition`	The definition which contain the parameters

Value

A character vector containing the roxygen tags

`get_default_parameters` *Get the default parameters of a method*

Description

Get the default parameters of a method

Usage

```
get_default_parameters(definition)
```

Arguments

`definition`	A TI method description

`get_ti_methods` *Return all TI that are installed in one or more packages*

Description

Return all TI that are installed in one or more packages

Usage

```
get_ti_methods(
  method_ids = NULL,
  as_tibble = TRUE,
  ti_packages = ifelse(is_dynmethods_installed(), "dynmethods", "dynwrap"),
  evaluate = FALSE
)
```

Arguments

`method_ids`	The method identifiers. NULL if listing all methods
`as_tibble`	Whether or not to return the ti_methods as a tibble
`ti_packages`	In which packages to look for TI methods. This will by default look into dynmethods if it is installed, otherwise in dynwrap.
`evaluate`	Whether to evaluate the functions

Value

A dataframe (or list if `as_tibble = FALSE`) containing the name (*id*) of the TI method and the function (*fun*) to load in the method.

Examples

```
head(get_ti_methods())
```

---

group_from_trajectory *Create a grouping from a trajectory*

Description

Grouping cells onto their edges, or grouping cells onto their nearest milestones

Usage

```
group_onto_trajectory_edges(trajectory, group_template = "{from}->{to}")

group_onto_nearest_milestones(trajectory)
```

Arguments

trajectory	The trajectory as created by `infer_trajectory()` or `add_trajectory()`
group_template	Processed by glue::glue to name the group

---

infer_trajectories *Infer one or more trajectories from a single-cell dataset*

Description

Infer one or more trajectories from a single-cell dataset

Usage

```
infer_trajectories(
  dataset,
  method,
  parameters = NULL,
  give_priors = NULL,
  seed = random_seed(),
  verbose = FALSE,
  return_verbose = FALSE,
  debug = FALSE,
  map_fun = map
)

infer_trajectory(
  dataset,
  method,
  parameters = NULL,
  give_priors = NULL,
  seed = random_seed(),
  verbose = FALSE,
  return_verbose = FALSE,
  debug = FALSE,
  ...
)
```

Arguments

`dataset`	One or more datasets as created by `wrap_data()` or `wrap_expression()`. Prior information can be added using `add_prior_information()`.
`method`	One or more methods. Must be one of: • an object or list of ti_... objects (e.g. `dynmethods::ti_comp1`, • a character vector containing the names of methods to execute (e.g. `"scorpius"`), • a character vector containing dockerhub repositories (e.g. `dynverse/paga`), or • a dynguidelines data frame.
`parameters`	A set of parameters to be used during trajectory inference. A parameter set must be a named list of parameters. If multiple methods were provided in the `method` parameter, `parameters` must be an unnamed list of the same length.
`give_priors`	All the priors a method is allowed to receive. Must be a subset of all available priors (priors).
`seed`	A seed to be passed to the TI method.
`verbose`	Whether or not to print information output.
`return_verbose`	Whether to store and return messages printed by the method.
`debug`	Used for debugging containers methods.
`map_fun`	A map function to use when inferring trajectories with multiple datasets or methods. Allows to parallellise the execution in an arbitrary way.
`...`	Any additional parameters given to the method, will be concatenated to the parameters argument

Value

`infer_trajectory`: A trajectory object, which is a list containing

- *milestone_ids*: The names of the milestones, a character vector.
- *milestone_network*: The network between the milestones, a dataframe with the from milestone, to milestone, length of the edge, and whether it is directed.
- *divergence_regions*: The regions between three or more milestones where cells are diverging, a dataframe with the divergence id, the milestone id and whether this milestone is the start of the divergence

- *milestone_percentages*: For each cell its closeness to a particular milestone, a dataframe with the cell id, the milestone id, and its percentage (a number between 0 and 1 where higher values indicate that a cell is close to the milestone).
- *progressions*: For each cell its progression along a particular edge of the *milestone_network*. Contains the same information as *milestone_percentages*. A dataframe with cell id, from milestone, to milestone, and its percentage (a number between 0 and 1 where higher values indicate that a cell is close to the 'to' milestone and far from the 'from' milestone).
- *cell_ids*: The names of the cells

Some methods will include additional information in the output, such as

- A dimensionality reduction (*dimred*), the location of the trajectory milestones and edges in this dimensionality reduction (*dimred_milestones*, *dimred_segment_progressions* and *dimred_segment_points*). See `add_dimred()` for more information on these objects.
- A cell grouping (*grouping*). See `add_grouping()` for more information on this object.

`infer_trajectories`: A tibble containing the dataset and method identifiers (*dataset_id* and *method_id*), the trajectory model as described above (*model*), and a *summary* containing the execution times, output and error if appropriate

Examples

```
dataset <- example_dataset
method <- get_ti_methods(as_tibble = FALSE)[[1]]$fun

trajectory <- infer_trajectory(dataset, method())

head(trajectory$milestone_network)
head(trajectory$progressions)
```

`label_milestones` *Label milestones either manually (*`label_milestones`*) or using marker genes (*`label_milestones_markers`*)*

Description

`label_milestones` can be used to manually assign labels to a milestone using their identifiers

Usage

```
label_milestones(trajectory, labelling)

label_milestones_markers(
  trajectory,
  markers,
  expression_source = "expression",
  n_nearest_cells = 20
)

is_wrapper_with_milestone_labelling(trajectory)

get_milestone_labelling(trajectory, label_milestones = NULL)
```

Arguments

`trajectory`	The trajectory as created by `infer_trajectory()` or `add_trajectory()`
`labelling`	Named character vector containing for a milestone a new label
`markers`	List containing for each label a list of marker genes
`expression_source`	The expression source
`n_nearest_cells`	The number of nearest cells to use for extracting milestone expression
`label_milestones`	How to label the milestones. Can be TRUE (in which case the labels within the trajectory will be used), "all" (in which case both given labels and milestone_ids will be used), a named character vector, or FALSE

Details

`label_milestones_markers` will assign a label to a milestone if its marker profile most closely resembles a given profile

Value

`label_milestones`: A trajectory object with *milestone_labelling*, a named vector where milestone identifiers are mapped to their labels

`get_milestone_labelling`: A named vector giving a mapping between milestones and their labels. If certain milestones were not given a label, this vector will give the identifiers themselves.

Examples

```
trajectory <- example_trajectory

# manual labelling
trajectory <- label_milestones(
  trajectory,
  labelling = c("milestone_begin" = "Let's go")
)
get_milestone_labelling(trajectory)

# marker gene labelling
trajectory <- label_milestones_markers(
  trajectory,
  markers = list(A_high = "A")
)
get_milestone_labelling(trajectory)
```

```
is_wrapper_with_milestone_labelling(trajectory)
```

`priors` *Metadata on priors*

Description

Metadata on priors

Usage

```
priors
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 13 rows and 6 columns.

Examples

```
priors
```

`prior_usages` *Metadata on prior usages*

Description

Metadata on prior usages

Usage

```
prior_usages
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 3 rows and 2 columns.

Examples

```
prior_usages
```

`project_trajectory` *Project a trajectory onto a dimensionality reduction*

Description

Project a trajectory onto a dimensionality reduction

Usage

```
project_trajectory(
  trajectory,
  dimred,
  waypoints = select_waypoints(trajectory),
  trajectory_projection_sd = sum(trajectory$milestone_network$length) * 0.05
)

project_milestones(
  trajectory,
  dimred,
  trajectory_projection_sd = sum(trajectory$milestone_network$length) * 0.05
)
```

Arguments

`trajectory`	The trajectory as created by `infer_trajectory()` or `add_trajectory()`
`dimred`	The dimensionality reduction of the cells. A matrix with the positions of cells (rows) in the dimensions (columns)
`waypoints`	A set of waypoints, which can be created by `select_waypoints()`. It is a list containing: • `waypoints`: a dataframe containing in the very least the waypoint_id • `milestone_percentages`: the positions of waypoints withing the trajectory • `geodesic_distances`: matrix with precalculated geodesic distances between waypoints (rows) and cells (columns), optional
`trajectory_projection_sd`	The standard deviation of the gaussian kernel

Value

A list containing

- *dimred_segment_points*: The dimensionality reduction of a set of points along the trajectory. A matrix with the position of points (rows) in the dimensions (columns)
- *dimred_segment_progressions* The progressions of the points. A dataframe containing the *from* and *to* milestones, and their *progression*. Has the same number of rows as *dimred_segment_points*

- *dimred_milestones*: The dimensionality reduction of the milestones. A matrix with the position of milestones (rows) in the dimensions (columns)

These objects can be given to `add_dimred()`

See Also

`add_dimred()`

`project_waypoints` *Project waypoints of a trajectory (e.g. milestones) into a space defined by cells (e.g. expression or a dimensionality reduction)*

Description

This will first calculate the geodesic distance of each cell to the waypoint. This distance is used as a weight

Usage

```
project_waypoints(
  trajectory,
  space,
  waypoints = select_waypoints(trajectory),
  trajectory_projection_sd = sum(trajectory$milestone_network$length) * 0.05
)
```

Arguments

`trajectory`	The trajectory as created by `infer_trajectory()` or `add_trajectory()`
`space`	A matrix with cells in rows and different dimensions in the columns. This is typically an expression matrix or a dimensionality reduction
`waypoints`	A set of waypoints, which can be created by `select_waypoints()`. It is a list containing: • `waypoints`: a dataframe containing in the very least the waypoint_id • `milestone_percentages`: the positions of waypoints withing the trajectory • `geodesic_distances`: matrix with precalculated geodesic distances between waypoints (rows) and cells (columns), optional
`trajectory_projection_sd`	The standard deviation of the gaussian kernel

Value

A matrix in which the waypoints (rows) were projected into a new space defined by the same number of dimensions (columns) as in the space argument

`random_seed` *Generate a random seed*

Description

From the current seed.

Usage

```
random_seed()
```

Value

A random seed

Examples

```
random_seed()
```

`select_waypoints` *Add or create waypoints to a trajectory*

Description

Waypoints are points along the trajectory, which do not necessarily correspond to cells. They are selected in such a way that all parts of the trajectory are covered

Usage

```
select_waypoints(
  trajectory,
  n_waypoints = 200,
  trafo = sqrt,
  resolution = sum(trafo(trajectory$milestone_network$length))/n_waypoints,
  recompute = FALSE
)

add_waypoints(
  trajectory,
  n_waypoints = 200,
  trafo = sqrt,
  resolution = sum(trafo(trajectory$milestone_network$length))/n_waypoints,
  recompute = FALSE
)

is_wrapper_with_waypoints(trajectory)
```

Arguments

`trajectory`	The trajectory as created by `infer_trajectory()` or `add_trajectory()`
`n_waypoints`	The number of waypoints
`trafo`	Transformation function of the edge lengths
`resolution`	The resolution of the waypoints, measured in the same units as the lengths of the milestone network edges, will be automatically computed using n_waypoints
`recompute`	Force recompute

Value

`add_waypoints` returns the trajectory with *waypoints* added, which is a list containing:

- *milestone_percentages* and *progressions*: The milestone percentages and progressions of each waypoint, in the same format as the cell equivalents (see `add_trajectory()`) but with a *waypoint_id* column instead of a *cell_id* column
- *geodesic_distances*: a matrix with the geodesic distance of each waypoint (rows) to every cell (columns)
- *waypoint_network*: a dataframe containing the network between consecutive waypoints, it contains information on the connected waypoints (*from* and *to*) and the edge on which they reside (*from_milestone_id* and *to_milestone_id*)
- *waypoints*: the waypoint identifiers

**`select_waypoints` returns the list as mentioned in `add_waypoints`

`simplify_igraph_network` *Simplify an igraph network such that consecutive linear edges are removed*

Description

- Nodes with degree 2 (or indegree 1 and outdegree 1) are removed: A -> B -> C becomes A -> C
- Cycles contain at least 3 nodes, ie. A -> B -> A becomes A -> B -> C -> A
- Loops are converted to a cycle, unless `allow_self_loops = TRUE`
- Duplicated edges are removed, unless `allow_duplcated_edges = FALSE`

Usage

```
simplify_igraph_network(
  gr,
  allow_duplicated_edges = TRUE,
  allow_self_loops = TRUE,
  force_keep = NULL,
  edge_points = NULL
)
```

Arguments

`gr`	An igraph object, see `igraph::graph()`
`allow_duplicated_edges`	Whether or not to allow duplicated edges between nodes.
`allow_self_loops`	Whether or not to allow self loops.
`force_keep`	Nodes that will not be removed under any condition
`edge_points`	Points that are on edges

Value

An igraph object, or a list with an igraph object and a data frame with edge points

Examples

```
net <- data.frame(
  from = 1:2,
  to = 2:3,
  length = 1,
  directed = TRUE,
  stringsAsFactors = F
)
gr <- igraph::graph_from_data_frame(net)
simplify_igraph_network(gr)

net <- data.frame(
  from = c(1, 2, 3, 1),
   to = c(2, 3, 1, 4),
    length = 1,
    directed = TRUE,
    stringsAsFactors = F
)
gr <- igraph::graph_from_data_frame(net)
simplify_igraph_network(gr)

net <- data.frame(
  from = c(1, 2, 3, 4),
   to = c(2, 3, 1, 5),
    length = 1,
    directed = TRUE,
    stringsAsFactors = F
)
gr <- igraph::graph_from_data_frame(net)
simplify_igraph_network(gr)
```

`simplify_trajectory` *Simplify a trajectory by removing transient milestones*

Description

- Milestones that are not a leaf or a branching point are removed: A -> B -> C becomes A -> C
- Cycles contain at least 3 nodes, ie. A -> B -> A becomes A -> B -> C -> A
- Loops are converted to a cycle, unless `allow_self_loops = TRUE`

Usage

```
simplify_trajectory(trajectory, allow_self_loops = FALSE)
```

Arguments

`trajectory`	The trajectory as created by `infer_trajectory()` or `add_trajectory()`
`allow_self_loops`	Whether or not to allow self loops.

Details

The positions of the cells within the trajectory remain the same.

Value

A trajectory object

`trajectory_types` *Metadata on the trajectory types*

Description

Metadata on the trajectory types

Usage

```
trajectory_types
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 9 rows and 6 columns.

Examples

```
trajectory_types
```

`trajectory_type_dag` *A DAG connecting different trajectory types*

Description

A DAG connecting different trajectory types

Usage

```
trajectory_type_dag
```

Format

An object of class `tbl_graph` (inherits from `igraph`) of length 9.

Examples

```
trajectory_type_dag
```

`wrapper_types` *Metadata on wrapper types*

Description

Metadata on wrapper types

Usage

```
wrapper_types
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 7 rows and 4 columns.

Examples

```
wrapper_types
```

`wrap_data` *A data wrapper for datasets and trajectories*

Description

A data wrapper for datasets and trajectories

Usage

```
wrap_data(
  id = NULL,
  cell_ids,
  cell_info = NULL,
  feature_ids = NULL,
  feature_info = NULL,
  ...
)

is_data_wrapper(dataset)
```

Arguments

`id`	A unique identifier for the data. If `NULL`, a random string will be generated.
`cell_ids`	The identifiers of the cells.
`cell_info`	Optional meta-information pertaining the cells.
`feature_ids`	The identifiers of the features.
`feature_info`	Optional meta-information pertaining the features.
`...`	Extra information to be stored in the wrapper.
`dataset`	A dataset created by `wrap_data()` or `wrap_expression()`

Value

A list containing *id*, *cell_ids* and *cell_info* (if specified)

Examples

```
dataset <- wrap_data(
  cell_ids = c("A", "B", "C")
)
dataset$cell_ids
```

`wrap_expression` *Create a wrapper object with expression and counts*

Description

Projected expression based on RNA velocity can also be added to the wrapper through the `expression_future` argument

Usage

```
wrap_expression(
  id = NULL,
  expression,
  counts,
  cell_info = NULL,
  feature_info = NULL,
  expression_future = NULL,
  ...
)
```

Arguments

`id`	A unique identifier for the data. If `NULL`, a random string will be generated.
`expression`	The normalised expression values of genes (columns) within cells (rows). This can be both a dense and sparse matrix.
`counts`	The counts values of genes (columns) within cells (rows). This can be both a dense and sparse matrix.
`cell_info`	Optional meta-information pertaining the cells.
`feature_info`	Optional meta-information of the features, a dataframe with at least *feature_id* as column
`expression_future`	Projected expression using RNA velocity of genes (columns) within cells (rows). This can be both a dense and sparse matrix.
`...`	extra information to be stored in the dataset

Details

Information about the cells and/or features can be added through `cell_info` and `feature_info`

Examples

```
dataset <- wrap_expression(
  counts = example_dataset$counts,
  expression = example_dataset$expression,
  expression_future = example_dataset$expression_future
)
```

```
dataset$counts[1:10, 1:3]
dataset$expression[1:10, 1:3]
dataset$expression_future[1:10, 1:3]
```

Index