Package ‘destiny’

February 20, 2026

Type Package

Title Creates diffusion maps

Version 3.24.0

Date 2014-12-19

Description Create and plot diffusion maps.

License GPL-3

URL https://theislab.github.io/destiny/,
https://github.com/theislab/destiny/,
https://www.helmholtz-muenchen.de/icb/destiny,
https://bioconductor.org/packages/destiny,
https://doi.org/10.1093/bioinformatics/btv715

BugReports https://github.com/theislab/destiny/issues

Encoding UTF-8

Depends R (>= 3.4.0)

Imports methods, graphics, grDevices, grid, utils, stats, Matrix, Rcpp (>= 0.10.3), RcppEigen, RSpectra (>= 0.14-0), irlba, pcaMethods, Biobase, BiocGenerics, SummarizedExperiment, SingleCellExperiment, ggplot2, ggplot.multistats, rlang, tidyr, tidyselect, ggthemes, VIM, knn.covertree, proxy, RcppHNSW, smoother, scales, scatterplot3d

LinkingTo Rcpp, RcppEigen, grDevices

SystemRequirements C++11

NeedsCompilation yes

Enhances rgl, SingleCellExperiment

Suggests knitr, rmarkdown, igraph, testthat, FNN, tidyverse, gridExtra, cowplot, conflicted, viridis, rgl, scRNAseq, org.Mm.eg.db, scran, repr

VignetteBuilder knitr

biocViews CellBiology, CellBasedAssays, Clustering, Software, Visualization

Collate 'RcppExports.R' 'aaa.r' 'accessor-generics.r' 'censoring.r' 'colorlegend.r' 'cube_helix.r' 'dataset-helpers.r' 'destiny-package.r' 's4-unions.r' 'dist-matrix-coerce.r' 'sigmas.r' 'diffusionmap.r' 'diffusionmap-methods-accession.r' 'diffusionmap-methods.r' 'plothelpers.r' 'diffusionmap-plotting.r' 'dpt-branching.r' 'dpt-helpers.r' 'dpt.r' 'dpt-methods-matrix.r' 'dpt-methods.r' 'utils.r' 'dpt-plotting.r' 'eig_decomp.r' 'expressionset-helpers.r' 'find_dm_k.r' 'gene-relevance.r' 'gene-relevance-methods.r' 'gene-relevance-plotting-differential-map.r' 'gene-relevance-plotting-gr-map.r' 'gene-relevance-plotting-rank.r' 'gene-relevance-plotting.r' 'guo-data.r' 'knn.r' 'l_which.r' 'methods-coercion.r' 'methods-extraction.r' 'methods-update.r' 'predict.r' 'projection-dist.r' 'rankcor.r' 'sigmas-plotting.r'

RoxygenNote 7.3.3

git_url https://git.bioconductor.org/packages/destiny

git_branch RELEASE_3_22

git_last_commit 9280aaa

git_last_commit_date 2025-11-06

Repository Bioconductor 3.22

Date/Publication 2026-02-20

Author Philipp Angerer [cre, aut] (ORCID: <https://orcid.org/0000-0002-0369-2888>),
Laleh Haghverdi [ctb],
Maren Büttner [ctb] (ORCID: <https://orcid.org/0000-0002-6189-3792>),
Fabian Theis [ctb] (ORCID: <https://orcid.org/0000-0002-2419-1943>),
Carsten Marr [ctb] (ORCID: <https://orcid.org/0000-0003-2154-4552>),
Florian Büttner [ctb] (ORCID: <https://orcid.org/0000-0001-5587-6761>)

Maintainer Philipp Angerer <phil.angerer@gmail.com>

Contents

`Coercion methods` *Coercion methods*

Description

Convert a `DiffusionMap` or `DPT` object to other classes

Usage

```
## S4 method for signature 'DiffusionMap'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

fortify.DiffusionMap(model, data, ...)

## S4 method for signature 'DPT'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

fortify.DPT(model, data, ...)

## S4 method for signature 'DPT'
as.matrix(x, ...)
```

Arguments

`x, model`	A `DiffusionMap` or `DPT` object
`row.names`	NULL or a character vector giving the row names for the data frame. Missing values are not allowed.
`optional`	logical. If TRUE, setting row names and converting column names (to syntactic names: see make.names) is optional.
`...`	Passed to `as.data.frame`
`data`	ignored

Details

fortify is a ggplot2 generic allowing a diffusion map to be used as `data` parameter in ggplot or qplot.

Value

An object of the desired class

See Also

DiffusionMap accession methods, Extraction methods, DiffusionMap methods for more

Examples

```
library(Biobase)
data(guo)
dm <- DiffusionMap(guo)
classes <- vapply(as.data.frame(dm), class, character(1L))
stopifnot(all(classes[paste0('DC', 1:20)] == 'numeric'))
stopifnot(all(classes[featureNames(guo) ] == 'numeric'))
stopifnot(all(classes[   varLabels(guo) ] == c('factor', 'integer')))
```

`colorlegend` *Color legend*

Description

Creates a color legend for a vector used to color a plot. It will use the current `palette()` or the specified `pal` as reference.

Usage

```
colorlegend(
  col,
  pal = palette(),
  log = FALSE,
  posx = c(0.9, 0.93),
  posy = c(0.05, 0.9),
  main = NULL,
  cex_main = par("cex.sub"),
  cex_axis = par("cex.axis"),
  col_main = par("col.sub"),
  col_lab = par("col.lab"),
  steps = 5,
  steps_color = 100,
  digit = 2,
  left = FALSE,
  ...,
  cex.main = NULL,
  cex.axis = NULL,
  col.main = NULL,
  col.lab = NULL
)
```

Arguments

`col`	Vector of factor, integer, or double used to determine the ticks.
`pal`	If `col` is double, pal is used as a continuous palette, else as categorical one
`log`	Use logarithmic scale?
`posx`	Left and right borders of the color bar relative to plot area (Vector of length 2; 0-1)
`posy`	Bottom and top borders of color bar relative to plot area (Vector of length 2; 0-1)
`main`	Legend title
`cex_main`	Size of legend title font (default: subtitle font size `par('cex.sub')`)
`cex_axis`	Size of ticks/category labels (default: axis font size `par('cex.axis')`)
`col_main`	Color of legend title (default: subtitle color `par('col.sub')`)
`col_lab`	Color of tick or category labels (default: axis color `par('col.lab')`)
`steps`	Number of labels in case of a continuous axis. If 0 or FALSE, draw no ticks
`steps_color`	Number of gradient samples in case of continuous axis
`digit`	Number of digits for continuous axis labels
`left`	logical. If TRUE, invert posx
`...`	Additional parameters for the text call used for labels
`cex.main`, `cex.axis`, `col.main`, `col.lab`	For compatibility with `par`

Details

When passed a factor or integer vector, it will create a discrete legend, whereas a double vector will result in a continuous bar.

Value

This function is called for the side effect of adding a colorbar to a plot and returns nothing/NULL.

Examples

```
color_data <- 1:6
par(mar = par('mar') + c(0, 0, 0, 3))
plot(sample(6), col = color_data)
colorlegend(color_data)
```

`cube_helix` *Sequential color palette using the cube helix system*

Description

Creates a perceptually monotonously decreasing (or increasing) lightness color palette with different tones. This was necessary in pre-viridis times, by now you can probably just use `hcl.colors`

Usage

```
cube_helix(
  n = 6,
  start = 0,
  r = 0.4,
  hue = 0.8,
  gamma = 1,
  light = 0.85,
  dark = 0.15,
  reverse = FALSE
)

scale_colour_cube_helix(
  ...,
  start = 0,
  r = 0.4,
  hue = 0.8,
  gamma = 1,
  light = 0.85,
  dark = 0.15,
  reverse = FALSE,
  discrete = TRUE,
  guide = if (discrete) "legend" else "colourbar"
)

scale_color_cube_helix(
  ...,
  start = 0,
  r = 0.4,
  hue = 0.8,
  gamma = 1,
  light = 0.85,
  dark = 0.15,
  reverse = FALSE,
  discrete = TRUE,
  guide = if (discrete) "legend" else "colourbar"
)

scale_fill_cube_helix(
  ...,
  start = 0,
  r = 0.4,
  hue = 0.8,
  gamma = 1,
  light = 0.85,
  dark = 0.15,
  reverse = FALSE,
  discrete = TRUE,
  guide = if (discrete) "legend" else "colourbar"
)
```

Arguments

`n`	Number of colors to return (default: 6)
`start`	Hue to start helix at (start $\in [0, 3]$, default: 0)
`r`	Number of rotations of the helix. Can be negative. (default: 0.4)
`hue`	Saturation. 0 means greyscale, 1 fully saturated colors (default: 0.8)
`gamma`	Emphasize darker (gamma < 1) or lighter (gamma > 1) colors (default: 1)
`light`	Lightest lightness (default: 0.85)
`dark`	Darkest lightness (default: 0.15)
`reverse`	logical. If TRUE, reverse lightness (default: FALSE)
`...`	parameters passed to `discrete_scale` or `continuous_scale`
`discrete`	If TRUE, return a discrete scale, if FALSE a continuous one (default: TRUE)
`guide`	Type of scale guide to use. See `guides`

Value

A `character` vector of hex colors with length `n`

Examples

```
palette(cube_helix())
image(matrix(1:6), col = 1:6, pch = 19, axes = FALSE)

cr <- scales::colour_ramp(cube_helix(12, r = 3))
r <- runif(100)
plot(1:100, r, col = cr(r), type = 'b', pch = 20)
```

`destiny` *Create and plot diffusion maps*

Description

The main function is `DiffusionMap`, which returns an object you can `plot` (`plot.DiffusionMap` is then called).

Author(s)

Maintainer: Philipp Angerer <`phil.angerer@gmail.com`> (ORCID)

Other contributors:

- Laleh Haghverdi <`laleh.haghverdi@helmholtz-muenchen.de`> [contributor]
- Maren Büttner <`maren.buettner@helmholtz-muenchen.de`> (ORCID) [contributor]
- Fabian Theis <`fabian.theis@helmholtz-muenchen.de`> (ORCID) [contributor]
- Carsten Marr <`carsten.marr@helmholtz-muenchen.de`> (ORCID) [contributor]
- Florian Büttner <`f.buettner@helmholtz-muenchen.de`> (ORCID) [contributor]

See Also

Useful links:

- https://theislab.github.io/destiny/
- https://github.com/theislab/destiny/
- https://www.helmholtz-muenchen.de/icb/destiny
- https://bioconductor.org/packages/destiny
- doi:10.1093/bioinformatics/btv715
- Report bugs at https://github.com/theislab/destiny/issues

Examples

```
demo(destiny, ask = FALSE)
```

destiny generics *destiny generics*

Description

destiny provides several generic methods and implements them for the `DiffusionMap` and `Sigmas` classes.

Usage

```
eigenvalues(object)

eigenvalues(object) <- value

eigenvectors(object)

eigenvectors(object) <- value

sigmas(object)

sigmas(object) <- value

dataset(object)

dataset(object) <- value

distance(object)

distance(object) <- value

optimal_sigma(object)
```

Arguments

`object`	Object from which to extract or to which to assign a value
`value`	Value to assign within an object

Value

`eigenvalues` retrieves the numeric eigenvalues

`eigenvectors` retrieves the eigenvectors matrix

`sigmas` retrieves the `Sigmas` from an object utilizing it as kernel width

`dataset` retrieves the data the object was created from

`distance` retrieves the distance metric used to create the object, e.g. `euclidean`

`optimal_sigma` retrieves the numeric value of the optimal sigma or local sigmas

See Also

DiffusionMap methods and Sigmas class for implementations

Examples

```
data(guo_norm)
dm <- DiffusionMap(guo_norm)
eigenvalues(dm)
eigenvectors(dm)
sigmas(dm)
optimal_sigma(dm)
dataset(dm)
distance(dm)
```

`DiffusionMap accession methods`

DiffusionMap accession methods

Description

Get and set eigenvalues, eigenvectors, and sigma(s) of a DiffusionMap object.

Usage

```
## S4 method for signature 'DiffusionMap'
eigenvalues(object)

## S4 replacement method for signature 'DiffusionMap'
eigenvalues(object) <- value

## S4 method for signature 'DiffusionMap'
eigenvectors(object)

## S4 replacement method for signature 'DiffusionMap'
eigenvectors(object) <- value

## S4 method for signature 'DiffusionMap'
sigmas(object)

## S4 replacement method for signature 'DiffusionMap'
```

```
sigmas(object) <- value

## S4 method for signature 'DiffusionMap'
dataset(object)

## S4 replacement method for signature 'DiffusionMap'
dataset(object) <- value

## S4 method for signature 'DiffusionMap'
distance(object)

## S4 replacement method for signature 'DiffusionMap'
distance(object) <- value

## S4 method for signature 'DiffusionMap'
optimal_sigma(object)
```

Arguments

`object`	A DiffusionMap
`value`	Vector of eigenvalues or matrix of eigenvectors to get/set

Value

The assigned or retrieved value

See Also

Extraction methods, DiffusionMap methods, Coercion methods for more

Examples

```
data(guo)
dm <- DiffusionMap(guo)
eigenvalues(dm)
eigenvectors(dm)
sigmas(dm)
dataset(dm)
optimal_sigma(dm)
```

`DiffusionMap methods` *DiffusionMap methods*

Description

Methods for external operations on diffusion maps

Usage

```
## S4 method for signature 'DiffusionMap'
print(x)

## S4 method for signature 'DiffusionMap'
show(object)
```

Arguments

`x, object`	A `DiffusionMap`

Value

The `DiffusionMap` object (`print`), or NULL (`show`), invisibly

See Also

DiffusionMap accession methods, Extraction methods, Coercion methods for more

Examples

```
data(guo)
dm <- DiffusionMap(guo)
print(dm)
show(dm)
```

`DiffusionMap-class` *Create a diffusion map of cells*

Description

The provided data can be a double matrix of expression data or a data.frame with all non-integer (double) columns being treated as expression data features (and the others ignored), an ExpressionSet, or a SingleCellExperiment.

Usage

```
DiffusionMap(
  data = stopifnot_distmatrix(distance),
  sigma = "local",
  k = find_dm_k(dataset_n_observations(data, distance) - 1L),
  n_eigs = min(20L, dataset_n_observations(data, distance) - 2L),
  density_norm = TRUE,
  ...,
  distance = c("euclidean", "cosine", "rankcor", "l2"),
  n_pcs = NULL,
  n_local = seq(to = min(k, 7L), length.out = min(k, 3L)),
  rotate = FALSE,
  censor_val = NULL,
  censor_range = NULL,
  missing_range = NULL,
  vars = NULL,
  knn_params = list(),
  verbose = !is.null(censor_range),
  suppress_dpt = FALSE
)
```

Arguments

`data`	Expression data to be analyzed and covariates. Provide `vars` to select specific columns other than the default: all double value columns. If `distance` is a distance matrix, `data` has to be a `data.frame` with covariates only.
`sigma`	Diffusion scale parameter of the Gaussian kernel. One of `'local'`, `'global'`, a (numeric) global sigma or a Sigmas object. When choosing `'global'`, a global sigma will be calculated using `find_sigmas`. (Optional. default: `'local'`) A larger sigma might be necessary if the eigenvalues can not be found because of a singularity in the matrix
`k`	Number of nearest neighbors to consider (default: a guess betweeen 100 and $n - 1$. See `find_dm_k`).
`n_eigs`	Number of eigenvectors/values to return (default: 20)
`density_norm`	logical. If TRUE, use density normalisation
`...`	Unused. All parameters to the right of the `...` have to be specified by name (e.g. `DiffusionMap(data, distance = 'cosine')`)
`distance`	Distance measurement method applied to `data` or a distance matrix/`dist`. For the allowed values, see `find_knn`. If this is a `sparseMatrix`, zeros are interpreted as "not a close neighbors", which allows the use of kNN-sparsified matrices (see the return value of `find_knn`.
`n_pcs`	Number of principal components to compute to base calculations on. Using e.g. 50 DCs results in more regular looking diffusion maps. The default NULL will not compute principal components, but use `reducedDims(data, 'pca')` if present. Set to NA to suppress using PCs.
`n_local`	If `sigma == 'local'`, the `n_local`th nearest neighbor(s) determine(s) the local sigma
`rotate`	logical. If TRUE, rotate the eigenvalues to get a slimmer diffusion map
`censor_val`	Value regarded as uncertain. Either a single value or one for every dimension (Optional, default: censor_val)
`censor_range`	Uncertainity range for censoring (Optional, default: none). A length-2-vector of certainty range start and end. TODO: also allow $2 \times G$ matrix
`missing_range`	Whole data range for missing value model. Has to be specified if NAs are in the data
`vars`	Variables (columns) of the data to use. Specifying NULL will select all columns (default: All floating point value columns)
`knn_params`	Parameters passed to `find_knn`
`verbose`	Show a progressbar and other progress information (default: do it if censoring is enabled)
`suppress_dpt`	Specify TRUE to skip calculation of necessary (but spacious) information for `DPT` in the returned object (default: FALSE)

Value

A DiffusionMap object:

Slots

`eigenvalues` Eigenvalues ranking the eigenvectors

`eigenvectors` Eigenvectors mapping the datapoints to `n_eigs` dimensions

`sigmas` Sigmas object with either information about the find_sigmas heuristic run or just local or optimal_sigma.

`data_env` Environment referencing the data used to create the diffusion map

`eigenvec0` First (constant) eigenvector not included as diffusion component.

`transitions` Transition probabilities. Can be NULL

`d` Density vector of transition probability matrix

`d_norm` Density vector of normalized transition probability matrix

`k` The k parameter for kNN

`n_pcs` Number of principal components used in kNN computation (NA if raw data was used)

`n_local` The `n_local`th nearest neighbor(s) is/are used to determine local kernel density

`density_norm` Was density normalization used?

`rotate` Were the eigenvectors rotated?

`distance` Distance measurement method used

`censor_val` Censoring value

`censor_range` Censoring range

`missing_range` Whole data range for missing value model

`vars` Vars parameter used to extract the part of the data used for diffusion map creation

`knn_params` Parameters passed to `find_knn`

See Also

DiffusionMap methods to get and set the slots. `find_sigmas` to pre-calculate a fitting global `sigma` parameter

Examples

```
data(guo)
DiffusionMap(guo)
DiffusionMap(guo, 13, censor_val = 15, censor_range = c(15, 40), verbose = TRUE)

covars <- data.frame(covar1 = letters[1:100])
dists <- dist(matrix(rnorm(100*10), 100))
DiffusionMap(covars, distance = dists)
```

dm_predict — *Predict new data points using an existing DiffusionMap. The resulting matrix can be used in the plot method for the DiffusionMap*

Description

Predict new data points using an existing DiffusionMap. The resulting matrix can be used in the plot method for the DiffusionMap

Usage

```
dm_predict(dm, new_data, ..., verbose = FALSE)
```

Arguments

`dm`	A `DiffusionMap` object.
`new_data`	New data points to project into the diffusion map. Can be a matrix, data.frame, ExpressionSet, or SingleCellExperiment.
`...`	Passed to `proxy::dist(new_data, data, dm@distance, ...)`.
`verbose`	Show progress messages?

Value

A $nrow(new\_data) \times ncol(eigenvectors(dif))$ matrix of projected diffusion components for the new data.

Examples

```
data(guo)
g1 <- guo[, guo$num_cells != 32L]
g2 <- guo[, guo$num_cells == 32L]
dm <- DiffusionMap(g1)
dc2 <- dm_predict(dm, g2)
plot(dm, new_dcs = dc2)
```

DPT matrix methods — *DPT Matrix methods*

Description

Treat DPT object as a matrix of cell-by-cell DPT distances.

Usage

```
## S4 method for signature 'DPT,index,index,logicalOrMissing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'DPT,index,missing,logicalOrMissing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'DPT,missing,index,logicalOrMissing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'DPT,missing,missing,logicalOrMissing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'DPT,index,index'
x[[i, j, ...]]

## S4 method for signature 'DPT'
nrow(x)

## S4 method for signature 'DPT'
ncol(x)

## S4 method for signature 'DPT'
dim(x)
```

Arguments

`x`	`DPT` object.
`i, j`	Numeric or logical index.
`...`	ignored
`drop`	If `TRUE`, coerce result to a vector if it would otherwise have 1 `%in% dim(result)`.

Value

`[` returns a dense matrix or (if applicable and `isTRUE(drop)`) a vector.

`[[` returns single distance value

`nrow` and `ncol` return the number of cells

`dim` returns `c(n_cells, n_cells)`

See Also

`as.matrix.DPT`

Examples

```
data(guo_norm)
dm <- DiffusionMap(guo_norm)
dpt <- DPT(dm)
set.seed(1)
plot(dpt[random_root(dpt), ], Biobase::exprs(guo_norm)['DppaI', ])
```

DPT methods *DPT methods*

Description

Methods for the DPT class. `branch_divide` subdivides branches for plotting (see the examples).

Usage

```
branch_divide(dpt, divide = integer(0L))

tips(dpt)

## S4 method for signature 'DPT'
dataset(object)

## S4 replacement method for signature 'DPT'
dataset(object) <- value
```

Arguments

`dpt, object`	DPT object
`divide`	Vector of branch numbers to use for division
`value`	Value of slot to set

Value

`branch_divide` and `dataset<-` return the changed object, `dataset` the extracted data, and `tips` the tip indices.

See Also

plot.DPT uses `branch_divide` for its `divide` argument.

Examples

```
data(guo_norm)
dpt <- DPT(DiffusionMap(guo_norm))
dpt_9_branches <- branch_divide(dpt, 1:3)
plot(dpt_9_branches, col_by = 'branch')
```

DPT-class *Diffusion Pseudo Time*

Description

Create pseudotime ordering and assigns cell to one of three branches

Usage

```
DPT(dm, tips = random_root(dm), ..., w_width = 0.1)
```

Arguments

`dm`	A `DiffusionMap` object. Its transition probabilities will be used to calculate the DPT
`tips`	The cell index/indices from which to calculate the DPT(s) (integer of length 1-3)
`...`	Unused. All parameters to the right of the `...` have to be specified by name (e.g. `DPT(dm, w_width = 0.2)`)
`w_width`	Window width to use for deciding the branch cutoff

Details

Treat it as a matrix of pseudotime by subsetting (`[` `dim` `nrow` `ncol` `as.matrix`), and as a list of pseudodime, and expression vectors (`$` `[[` `names` `as.data.frame`).

Value

A `DPT` object:

Slots

`branch` `matrix` (of `integer`) recursive branch labels for each cell (row); `NA` for undeceided. Use `branch_divide` to modify this.

`tips` `matrix` (of `logical`) indicating if a cell (row) is a tip of the corresponding banch level (col)

`dm` `DiffusionMap` used to create this DPT object

Examples

```
data(guo_norm)
dm <- DiffusionMap(guo_norm)
dpt <- DPT(dm)
str(dpt)
```

eig_decomp — *Fast eigen decomposition using* `eigs`

Description

By default uses a random initialization vector that you can make deterministic using `set.seed` or override by specifying `opts = list(initvec = ...)`.

Usage

```
eig_decomp(m, n_eigs, sym, ..., opts = list())
```

Arguments

`m`	A matrix (e.g. from the Matrix package) or a function (see `eigs`).
`n_eigs`	Number of eigenvectors to return.
`sym`	defunct and ignored.
`...`	Passed to `eigs`.
`opts`	Passed to `eigs`.

Value

see `eigs`.

Examples

```
eig_decomp(cbind(c(1,0,-1), c(0,1,0), c(-1,0,1)), 2)
```

ExpressionSet helper methods — *Convert object to ExpressionSet or read it from a file*

Description

These functions present quick way to create ExpressionSet objects.

Usage

```
as.ExpressionSet(x, ...)

## S4 method for signature 'data.frame'
as.ExpressionSet(x, annotation_cols = !sapply(x, is.double))

read.ExpressionSet(file, header = TRUE, ...)
```

Arguments

`x`	data.frame to convert to an ExpressionSet.
`...`	Additional parameters to read.table
`annotation_cols`	The data.frame columns used as annotations. All others are used as expressions. (Logical, character or numerical index array)
`file`	File path to read ASCII data from
`header`	Specifies if the file has a header row.

Details

They work by using all continuous (double) columns as expression data, and all others as observation annotations.

Value

an ExpressionSet object

See Also

read.table on which `read.ExpressionSet` is based, and ExpressionSet.

Examples

```
library(Biobase)
df <- data.frame(Time  = seq_len(3), #integer column
                 Actb  = c(0.05, 0.3, 0.8),
                 Gapdh = c(0.2, 0.03, 0.1))
set <- as.ExpressionSet(df)
rownames(exprs(set)) == c('Actb', 'Gapdh')
phenoData(set)$Time == 1:3
```

`Extraction methods` *Extraction methods*

Description

Extract common information from objects. Apart from the input data's branches, you can extract diffusion components via `$DCx`. From `DPT` objects, you can also extract the branch label via `$Branch`, or the diffusion pseudo time for a numbered cell via `$DPTx`.

Usage

```
## S4 method for signature 'DiffusionMap'
names(x)

## S4 method for signature 'DPT'
names(x)

## S4 method for signature 'DiffusionMap,character,missing'
```

```
x[[i, j, ...]]

## S4 method for signature 'DPT,character,missing'
x[[i, j, ...]]

## S4 method for signature 'DiffusionMap'
x$name

## S4 method for signature 'DPT'
x$name
```

Arguments

`x`	`DiffusionMap` or `DPT` object
`i, name`	Name of a diffusion component `'DCx'`, `'DPTx'`, `'Branch'` or column from the data
`j`	N/A
`...`	ignored

Value

The names or data row, see respective generics.

See Also

Extract, `names` for the generics. DiffusionMap accession methods, DiffusionMap methods, Coercion methods for more

Examples

```
data(guo)
dm <- DiffusionMap(guo)
dm$DC1        # A diffusion component
dm$Actb       # A gene expression vector
dm$num_cells  # Phenotype metadata

dpt <- DPT(dm)
dm$Branch
dm$DPT1
```

`find_dm_k` *Find a suitable k*

Description

The `k` parameter for the `k` nearest neighbors used in DiffusionMap should be as big as possible while still being computationally feasible. This function approximates it depending on the size of the dataset `n`.

Usage

```
find_dm_k(n, min_k = 100L, small = 1000L, big = 10000L)
```

Arguments

`n`	Number of possible neighbors (nrow(dataset) - 1)
`min_k`	Minimum number of neighbors. Will be chosen for $n \geq big$
`small`	Number of neighbors considered small. If/where $n \leq small$, n itself will be returned.
`big`	Number of neighbors considered big. If/where $n \geq big$, `min_k` will be returned.

Value

A vector of the same length as `n` that contains suitable `k` values for the respective `n`

Examples

```
curve(find_dm_k(n),     0, 13000, xname = 'n')
curve(find_dm_k(n) / n, 0, 13000, xname = 'n')
```

`find_knn` *kNN search*

Description

Approximate k nearest neighbor search with flexible distance function.

Usage

```
find_knn(
  data,
  k,
  ...,
  query = NULL,
  distance = c("euclidean", "cosine", "rankcor", "l2"),
  method = c("covertree", "hnsw"),
  sym = TRUE,
  verbose = FALSE
)
```

Arguments

`data`	Data matrix
`k`	Number of nearest neighbors
`...`	Parameters passed to `hnsw_knn`
`query`	Query matrix. Leave it out to use `data` as query
`distance`	Distance metric to use. Allowed measures: Euclidean distance (default), cosine distance $(1 - corr(c_1, c_2))$ or rank correlation distance $(1 - corr(rank(c_1), rank(c_2)))$
`method`	Method to use. `'hnsw'` is tunable with `...` but generally less exact than `'covertree'` (default: 'covertree')
`sym`	Return a symmetric matrix (as long as query is NULL)?
`verbose`	Show a progressbar? (default: FALSE)

Value

A `list` with the entries:

`index` A $nrow(data) \times k$ integer matrix containing the indices of the k nearest neighbors for each cell.

`dist` A $nrow(data) \times k$ double matrix containing the distances to the k nearest neighbors for each cell.

`dist_mat` A `dgCMatrix` if `sym == TRUE`, else a `dsCMatrix` ($nrow(query) \times nrow(data)$). Any zero in the matrix (except for the diagonal) indicates that the cells in the corresponding pair are close neighbors.

`find_sigmas` *Calculate the average dimensionality for m different gaussian kernel widths (σ).*

Description

The sigma with the maximum value in average dimensionality is close to the ideal one. Increasing step number gets this nearer to the ideal one.

Usage

```
find_sigmas(
  data,
  step_size = 0.1,
  steps = 10L,
  start = NULL,
  sample_rows = 500L,
  early_exit = FALSE,
  ...,
  censor_val = NULL,
  censor_range = NULL,
  missing_range = NULL,
  vars = NULL,
  verbose = TRUE
)
```

Arguments

`data`	Data set with n observations. Can be a data.frame, matrix, ExpressionSet or SingleCellExperiment.
`step_size`	Size of log-sigma steps
`steps`	Number of steps/calculations
`start`	Initial value to search from. (Optional. default: $\log_{10}(min(dist(data)))$)
`sample_rows`	Number of random rows to use for sigma estimation or vector of row indices/names to use. In the first case, only used if actually smaller than the number of available rows (Optional. default: 500)
`early_exit`	logical. If TRUE, return if the first local maximum is found, else keep running

`...`	Unused. All parameters to the right of the ... have to be specified by name (e.g. `find_sigmas(data, verbose = FALSE)`)
`censor_val`	Value regarded as uncertain. Either a single value or one for every dimension
`censor_range`	Uncertainity range for censoring. A length-2-vector of certainty range start and end. TODO: also allow $2 \times G$ matrix
`missing_range`	Whole data range for missing value model. Has to be specified if NAs are in the data
`vars`	Variables (columns) of the data to use. Specifying TRUE will select all columns (default: All floating point value columns)
`verbose`	logical. If TRUE, show a progress bar and plot the output

Value

Object of class Sigmas

See Also

`Sigmas`, the class returned by this; `DiffusionMap`, the class this is used for

Examples

```
data(guo)
sigs <- find_sigmas(guo, verbose = TRUE)
DiffusionMap(guo, sigs)
```

`find_tips` *Find tips in a DiffusionMap object*

Description

Find tips in a DiffusionMap object

Usage

```
find_tips(dm_or_dpt, root = random_root(dm_or_dpt))
```

Arguments

`dm_or_dpt`	A `DiffusionMap` or `DPT` object
`root`	Root cell index from which to find tips. (default: random)

Value

An integer vector of length 3

Examples

```
data(guo)
dm <- DiffusionMap(guo)
is_tip <- l_which(find_tips(dm), len = ncol(guo))
plot(dm, col = factor(is_tip))
```

`Gene Relevance methods`

Gene Relevance methods

Description

`featureNames <- ...` can be used to set the gene names used for plotting (e.g. if the data contains hardly readably gene or transcript IDs). `dataset` gets the expressions used for the gene relevance calculations, and `distance` the distance measure.

Usage

```
## S4 method for signature 'GeneRelevance'
print(x)

## S4 method for signature 'GeneRelevance'
show(object)

## S4 method for signature 'GeneRelevance'
featureNames(object)

## S4 replacement method for signature 'GeneRelevance,characterOrFactor'
featureNames(object) <- value

## S4 method for signature 'GeneRelevance'
dataset(object)

## S4 replacement method for signature 'GeneRelevance'
dataset(object) <- value

## S4 method for signature 'GeneRelevance'
distance(object)

## S4 replacement method for signature 'GeneRelevance'
distance(object) <- value
```

Arguments

`x, object`	GeneRelevance object
`value`	A text vector (`character` or `factor`)

Value

`dataset`, `distance`, and `featureNames` return the stored properties. The other methods return a GeneRelevance object (`print, ... <- ...`), or NULL (`show`), invisibly

See Also

`gene_relevance`, Gene Relevance plotting

Examples

```
data(guo_norm)
dm <- DiffusionMap(guo_norm)
gr <- gene_relevance(dm)
stopifnot(distance(gr) == distance(dm))
featureNames(gr)[[37]] <- 'Id2 (suppresses differentiation)'
# now plot it with the changed gene name(s)
```

GeneRelevance-class *Gene relevances for entire data set*

Description

The relevance map is cached insided of the DiffusionMap.

Usage

```
gene_relevance(
  coords,
  exprs,
  ...,
  k = 20L,
  dims = 1:2,
  distance = NULL,
  smooth = TRUE,
  remove_outliers = FALSE,
  verbose = FALSE
)

## S4 method for signature 'DiffusionMap,missing'
gene_relevance(
  coords,
  exprs,
  ...,
  k = 20L,
  dims = 1:2,
  distance = NULL,
  smooth = TRUE,
  remove_outliers = FALSE,
  verbose = FALSE
)

## S4 method for signature 'matrix,dMatrixOrMatrix'
gene_relevance(
  coords,
  exprs,
  ...,
  pcs = NULL,
  knn_params = list(),
  weights = 1,
  k,
  dims,
  distance,
  smooth,
  remove_outliers,
  verbose
)
```

Arguments

coords	A DiffusionMap object or a cells $\times$ dims matrix.
exprs	An cells $\times$ genes matrix. Only provide if coords is no DiffusionMap.
...	Unused. All parameters to the right of the ... have to be specified by name.
k	Number of nearest neighbors to use
dims	Index into columns of coord
distance	Distance measure to use for the nearest neighbor search.
smooth	Smoothing parameters c(window, alpha) (see smth.gaussian). Alternatively TRUE to use the smoother defaults or FALSE to skip smoothing,
remove_outliers	Remove cells that are only within one other cell's nearest neighbor, as they tend to get large norms.
verbose	If TRUE, log additional info to the console
pcs	A cell $\times$ n_pcs matrix of principal components to use for the distances.
knn_params	A list of parameters for find_knn.
weights	Weights for the partial derivatives. A vector of the same length as dims.

Value

A GeneRelevance object:

Slots

coords A cells $\times$ dims matrix or sparseMatrix of coordinates (e.g. diffusion components), reduced to the dimensions passed as dims

exprs A cells $\times$ genes matrix of expressions

partials Array of partial derivatives wrt to considered dimensions in reduced space (genes $\times$ cells $\times$ dimensions)

partials_norm Matrix with norm of aforementioned derivatives. (n_genes $\times$ cells)

nn_index Matrix of k nearest neighbor indices. (cells $\times$ k)

dims Column index for plotted dimensions. Can character, numeric or logical

distance Distance measure used in the nearest neighbor search. See find_knn

smooth_window Smoothing window used (see smth.gaussian)

smooth_alpha Smoothing kernel width used (see smth.gaussian)

See Also

Gene Relevance methods, Gene Relevance plotting: plot_differential_map/plot_gene_relevance

Examples

```
data(guo_norm)
dm <- DiffusionMap(guo_norm)
gr <- gene_relevance(dm)

m <- t(Biobase::exprs(guo_norm))
gr_pca <- gene_relevance(prcomp(m)$x, m)
# now plot them!
```

---

guo *Guo at al. mouse embryonic stem cell qPCR data*

Description

Gene expression data of 48 genes and an annotation column `$num_cells` containing the cell stage at which the embryos were harvested.

Usage

```
data(guo)
data(guo_norm)
```

Format

An ExpressionSet with 48 features, 428 observations and 2 phenoData annotations.

Details

The data is normalized using the mean of two housekeeping genes. The difference between guo and `guo_norm` is the LoD being set to 10 in the former, making it usable with the `censor_val` parameter of DiffusionMap.

Value

an ExpressionSet with 48 features and 428 observations containing qPCR Ct values and a "num.cells" observation annotation.

Author(s)

Guoji Guo, Mikael Huss, Guo Qing Tong, Chaoyang Wang, Li Li Sun, Neil D. Clarke, Paul Robson `<robsonp@gis.a-star.edu.sg>`

References

`http://www.sciencedirect.com/science/article/pii/S1534580710001103`

`l_which` *Logical which*

Description

Inverse of which. Converts an array of numeric or character indices to a logical index array. This function is useful if you need to perform logical operation on an index array but are only given numeric indices.

Usage

```
l_which(idx, nms = seq_len(len), len = length(nms), useNames = TRUE)
```

Arguments

`idx`	Numeric or character indices.
`nms`	Array of names or a sequence. Required if `idx` is a character array
`len`	Length of output array. Alternative to `nms` if `idx` is numeric
`useNames`	Use the names of nms or idx

Details

Either `nms` or `len` has to be specified.

Value

Logical vector of length `len` or the same length as `nms`

Examples

```
all(l_which(2, len = 3L) == c(FALSE, TRUE, FALSE))
all(l_which(c('a', 'c'), letters[1:3]) == c(TRUE, FALSE, TRUE))
```

`plot.DiffusionMap` *3D or 2D plot of diffusion map*

Description

If you want to plot the eigenvalues, simply `plot(eigenvalues(dm)[start:end], ...)`

Usage

```
plot.DiffusionMap(
  x,
  dims = 1:3,
  new_dcs = if (!is.null(new_data)) dm_predict(x, new_data),
  new_data = NULL,
  col = NULL,
  col_by = NULL,
  col_limits = NULL,
  col_new = "red",
  pal = NULL,
  pal_new = NULL,
  ...,
  ticks = FALSE,
  axes = TRUE,
  box = FALSE,
  legend_main = col_by,
  legend_opts = list(),
  interactive = FALSE,
  draw_legend = !is.null(col_by) || (length(col) > 1 && !is.character(col)),
  consec_col = TRUE,
  col_na = "grey",
  plot_more = function(p, ..., rescale = NULL) p
)

## S4 method for signature 'DiffusionMap,numeric'
plot(x, y, ...)

## S4 method for signature 'DiffusionMap,missing'
plot(x, y, ...)
```

Arguments

`x`	A DiffusionMap
`dims, y`	Diffusion components (eigenvectors) to plot (default: first three components; 1:3)
`new_dcs`	An optional matrix also containing the rows specified with `y` and plotted. (default: no more points)
`new_data`	A data set in the same format as `x` that is used to create `new_dcs <- dm_predict(dif, new_data)`
`col`	Single color string or vector of discrete or categoric values to be mapped to colors. E.g. a column of the data matrix used for creation of the diffusion map. (default: `cluster_louvain` if igraph is installed)
`col_by`	Specify a `dataset(x)` or `phenoData(dataset(x))` column to use as color
`col_limits`	If `col` is a continuous (=double) vector, this can be overridden to map the color range differently than from min to max (e.g. specify `c(0, 1)`)
`col_new`	If `new_dcs` is given, it will take on this color. A vector is also possible. (default: red)
`pal`	Palette used to map the `col` vector to colors. (default: use `hcl.colors` for continuous and `palette()` for discrete data)

`pal_new`	Palette used to map the `col_new` vector to colors. (default: see `pal` argument)
`...`	Parameters passed to plot, scatterplot3d, or plot3d (if `interactive == TRUE`)
`ticks`	logical. If TRUE, show axis ticks (default: FALSE)
`axes`	logical. If TRUE, draw plot axes (default: Only if `ticks` is TRUE)
`box`	logical. If TRUE, draw plot frame (default: TRUE or the same as `axes` if specified)
`legend_main`	Title of legend. (default: nothing unless `col_by` is given)
`legend_opts`	Other colorlegend options (default: empty list)
`interactive`	Use plot3d to plot instead of scatterplot3d?
`draw_legend`	logical. If TRUE, draw color legend (default: TRUE if `col_by` is given or `col` is given and a vector to be mapped)
`consec_col`	If `col` or `col_by` refers to an integer column, with gaps (e.g. `c(5,0,0,3)`) use the palette color consecutively (e.g. `c(3,1,1,2)`)
`col_na`	Color for `NA` in the data. specify `NA` to hide.
`plot_more`	Function that will be called while the plot margins are temporarily changed (its `p` argument is the rgl or scatterplot3d instance or NULL, its `rescale` argument is `NULL`, a `list(from = c(a, b), to = c(c, d))`), or an array of shape $from\|to \times dims \times min\|max$, i.e. $2 \times length(dims) \times 2$. In case of 2d plotting, it should take and return a ggplot2 object.

Details

If you specify negative numbers as diffusion components (e.g. `plot(dm, c(-1,2))`), then the corresponding components will be flipped.

Value

The return value of the underlying call is returned, i.e. a scatterplot3d or rgl object.

Examples

```
data(guo)
plot(DiffusionMap(guo))
```

`plot.DPT` *Plot DPT*

Description

Plots diffusion components from a Diffusion Map and the accompanying Diffusion Pseudo Time (`DPT`)

Usage

```
plot.DPT(
  x,
  root = NULL,
  paths_to = integer(0L),
  dcs = 1:2,
  divide = integer(0L),
  w_width = 0.1,
  col_by = "dpt",
  col_path = rev(palette()),
  col_tip = "red",
  ...,
  col = NULL,
  legend_main = col_by
)

## S4 method for signature 'DPT,numeric'
plot(x, y, ...)

## S4 method for signature 'DPT,missing'
plot(x, y, ...)
```

Arguments

`x`	A `DPT` object.
`paths_to`	Numeric Branch IDs. Are used as target(s) for the path(s) to draw.
`dcs`	The dimensions to use from the DiffusionMap
`divide`	If `col_by = 'branch'`, this specifies which branches to divide. (see `branch_divide`)
`w_width`	Window width for smoothing the path (see `smth.gaussian`)
`col_by`	Color by 'dpt' (DPT starting at `branches[[1]]`), 'branch', or a veriable of the data.
`col_path`	Colors for the path or a function creating n colors
`col_tip`	Color for branch tips
`...`	Graphical parameters supplied to `plot.DiffusionMap`
`col`	See `plot.DiffusionMap`. This overrides `col_by`
`legend_main`	See `plot.DiffusionMap`.
`y, root`	Root branch ID. Will be used as the start of the DPT. (default: lowest branch ID) (If longer than size 1, will be interpreted as `c(root, branches)`)

Value

The return value of the underlying call is returned, i.e. a scatterplot3d or rgl object for 3D plots.

Examples

```
data(guo_norm)
dm <- DiffusionMap(guo_norm)
dpt <- DPT(dm)
plot(dpt)
plot(dpt, 2L,      col_by = 'branch')
plot(dpt, 1L, 2:3, col_by = 'num_cells')
plot(dpt, col_by = 'DPT3')
```

`plot.Sigmas` *Plot Sigmas object*

Description

Plot Sigmas object

Usage

```
## S4 method for signature 'Sigmas,missing'
plot(
  x,
  col = par("fg"),
  col_highlight = "#E41A1C",
  col_line = "#999999",
  type = c("b", "b"),
  pch = c(par("pch"), 4L),
  only_dim = FALSE,
  ...,
  xlab = NULL,
  ylab = NULL,
  main = ""
)
```

Arguments

`x`	Sigmas object to plot
`col`	Vector of bar colors or single color for all bars
`col_highlight`	Color for highest bar. Overrides col
`col_line`	Color for the line and its axis
`type`	Plot type of both lines. Can be a vector of length 2 to specify both separately (default: 'b' aka "both lines and points")
`pch`	Point identifier for both lines. Can be a vector of length 2 to specify both separately (default: `par(pch)` and 4 (a '$\times$'))
`only_dim`	logical. If TRUE, only plot the derivative line
`...`	Options passed to the call to plot
`xlab`	X label. NULL to use default
`ylab`	Either one y label or y labels for both plots. NULL to use both defauts, a NULL in a list of length 2 to use one default.
`main`	Title of the plot

Value

This method plots a Sigma object to the current device and returns nothing/NULL

Examples

```
data(guo)
sigs <- find_sigmas(guo)
plot(sigs)
```

---

plot_differential_map *Plot gene relevance or differential map*

Description

`plot(gene_relevance, 'Gene')` plots the differential map of this/these gene(s), `plot(gene_relevance)` a relevance map of a selection of genes. Alternatively, you can use `plot_differential_map` or `plot_gene_relevance` on a `GeneRelevance` or `DiffusionMap` object, or with two matrices.

Usage

```
plot_differential_map(
  coords,
  exprs,
  ...,
  genes,
  dims = 1:2,
  pal = hcl.colors,
  faceter = facet_wrap(~Gene)
)

## S4 method for signature 'matrix,matrix'
plot_differential_map(
  coords,
  exprs,
  ...,
  genes,
  dims = 1:2,
  pal = hcl.colors,
  faceter = facet_wrap(~Gene)
)

## S4 method for signature 'DiffusionMap,missing'
plot_differential_map(
  coords,
  exprs,
  ...,
  genes,
  dims = 1:2,
  pal = hcl.colors,
  faceter = facet_wrap(~Gene)
)

## S4 method for signature 'GeneRelevance,missing'
plot_differential_map(
  coords,
  exprs,
  ...,
  genes,
  dims = 1:2,
  pal = hcl.colors,
  faceter = facet_wrap(~Gene)
)

plot_gene_relevance(
  coords,
  exprs,
  ...,
  iter_smooth = 2L,
  n_top = 10L,
  genes = NULL,
  dims = 1:2,
  pal = palette(),
  col_na = "grey",
  limit = TRUE
)

## S4 method for signature 'matrix,matrix'
plot_gene_relevance(
  coords,
  exprs,
  ...,
  iter_smooth = 2L,
  n_top = 10L,
  genes = NULL,
  dims = 1:2,
  pal = palette(),
  col_na = "grey",
  limit = TRUE
)

## S4 method for signature 'DiffusionMap,missing'
plot_gene_relevance(
  coords,
  exprs,
  ...,
  iter_smooth = 2L,
  n_top = 10L,
  genes = NULL,
  dims = 1:2,
  pal = palette(),
  col_na = "grey",
  limit = TRUE
)

## S4 method for signature 'GeneRelevance,missing'
plot_gene_relevance(
  coords,
  exprs,
  ...,
  iter_smooth = 2L,
  n_top = 10L,
  genes = NULL,
  dims = 1:2,
  pal = palette(),
  col_na = "grey",
  limit = TRUE
)

plot_gene_relevance_rank(
  coords,
  exprs,
  ...,
  genes,
  dims = 1:2,
  n_top = 10L,
  pal = c("#3B99B1", "#F5191C"),
  bins = 10L,
  faceter = facet_wrap(~Gene)
)

## S4 method for signature 'matrix,matrix'
plot_gene_relevance_rank(
  coords,
  exprs,
  ...,
  genes,
  dims = 1:2,
  n_top = 10L,
  pal = c("#3B99B1", "#F5191C"),
  bins = 10L,
  faceter = facet_wrap(~Gene)
)

## S4 method for signature 'DiffusionMap,missing'
plot_gene_relevance_rank(
  coords,
  exprs,
  ...,
  genes,
  dims = 1:2,
  n_top = 10L,
  pal = c("#3B99B1", "#F5191C"),
  bins = 10L,
  faceter = facet_wrap(~Gene)
)

## S4 method for signature 'GeneRelevance,missing'
plot_gene_relevance_rank(
  coords,
  exprs,
  ...,
  genes,
  dims = 1:2,
  n_top = 10L,
  pal = c("#3B99B1", "#F5191C"),
  bins = 10L,
  faceter = facet_wrap(~Gene)
)

## S4 method for signature 'GeneRelevance,character'
plot(x, y, ...)

## S4 method for signature 'GeneRelevance,numeric'
plot(x, y, ...)

## S4 method for signature 'GeneRelevance,missing'
plot(x, y, ...)
```

Arguments

`coords`	A `DiffusionMap`/`GeneRelevance` object or a cells $\times$ dims `matrix`.
`exprs`	An cells $\times$ genes `matrix`. Only provide if coords is a matrix.
`...`	Passed to `plot_differential_map`/`plot_gene_relevance`.
`genes`	Genes to base relevance map on (vector of strings). You can also pass an index into the gene names (vector of numbers or logicals with length > 1). The default NULL means all genes.
`dims`	Names or indices of dimensions to plot. When not plotting a `GeneRelevance` object, the relevance for the dimensions `1:max(dims)` will be calculated.
`pal`	Palette. Either A colormap function or a list of colors.
`faceter`	A ggplot faceter like `facet_wrap(~ Gene)`.
`iter_smooth`	Number of label smoothing iterations to perform on relevance map. The higher the more homogenous and the less local structure.
`n_top`	Number the top n genes per cell count towards the score defining which genes to return and plot in the relevance map.
`col_na`	Color for cells that end up with no most relevant gene.
`limit`	Limit the amount of displayed gene labels to the amount of available colors in `pal`?
`bins`	Number of hexagonal bins for `plot_gene_relevance_rank`.
`x`	`GeneRelevance` object.
`y`	Gene name(s) or index/indices to create differential map for. (integer or character)

Value

ggplot2 plot, when plotting a relevance map with a list member `$ids` containing the gene IDs used.

See Also

gene_relevance, Gene Relevance methods

Examples

```
data(guo_norm)
dm <- DiffusionMap(guo_norm)
gr <- gene_relevance(dm)
plot(gr)          # or plot_gene_relevance(dm)
plot(gr, 'Fgf4')  # or plot_differential_map(dm, 'Fgf4')

guo_norm_mat <- t(Biobase::exprs(guo_norm))
pca <- prcomp(guo_norm_mat)$x
plot_gene_relevance(pca, guo_norm_mat, dims = 2:3)
plot_differential_map(pca, guo_norm_mat, genes = c('Fgf4', 'Nanog'))
```

---

projection_dist   *Projection distance*

Description

Projection distance

Usage

```
projection_dist(dm, new_dcs = NULL, ..., new_data, verbose = FALSE)
```

Arguments

dm	A DiffusionMap object.
new_dcs	Diffusion component matrix of which to calculate the distance to the data.
...	Passed to proxy::dist if new_data was passed.
new_data	New data points to project into the diffusion map. Can be a matrix, data.frame, ExpressionSet, or SingleCellExperiment.
verbose	If TRUE, log additional info to the console.

Value

A vector of distances each new data point has to the existing data.

Examples

```
data(guo_norm)
g2_32 <- guo_norm[, guo_norm$num_cells < 64]
g64  <- guo_norm[, guo_norm$num_cells == 64]
dm <- DiffusionMap(g2_32)
d <- projection_dist(dm, new_data = g64)
```

random_root *Find a random root cell index*

Description

Finds a cell that has the maximum DPT distance from a randomly selected one.

Usage

```
random_root(dm_or_dpt)
```

Arguments

dm_or_dpt	A DiffusionMap or DPT object

Value

A cell index

Examples

```
data(guo)
dm <- DiffusionMap(guo)
random_root(dm)
```

Sigmas-class *Sigmas Object*

Description

Holds the information about how the sigma parameter for a DiffusionMap was obtained, and in this way provides a plotting function for the find_sigmas heuristic. You should not need to create a Sigmas object yourself. Provide sigma to DiffusionMap instead or use find_sigmas.

Usage

```
Sigmas(...)

## S4 method for signature 'Sigmas'
optimal_sigma(object)

## S4 method for signature 'Sigmas'
print(x)

## S4 method for signature 'Sigmas'
show(object)
```

Arguments

`object, x`	Sigmas object
`...`	See “**Slots**” below

Details

A Sigmas object is either created by find_sigmas or by specifying the `sigma` parameter to DiffusionMap.

In the second case, if the `sigma` parameter is just a number, the resulting `Sigmas` object has all slots except of `optimal_sigma` set to `NULL`.

Value

`Sigmas` creates an object of the same class

`optimal_sigma` retrieves the numeric value of the optimal sigma or local sigmas

Slots

`log_sigmas` Vector of length m containing the $\log_{10}$ of the σs

`dim_norms` Vector of length $m - 1$ containing the average dimensionality $\langle p \rangle$ for the respective kernel widths

`optimal_sigma` Multiple local sigmas or the mean of the two global σs around the highest $\langle p \rangle$ (`c(optimal_idx, optimal_idx+1L)`)

`optimal_idx` The index of the highest $\langle p \rangle$.

`avrd_norms` Vector of length m containing the average dimensionality for the corresponding sigma.

See Also

`find_sigmas`, the function to determine a locally optimal sigma and returning this class

Examples

```
data(guo)
sigs <- find_sigmas(guo, verbose = FALSE)
optimal_sigma(sigs)
print(sigs)
```

`updateObject methods` *Update old destiny objects to a newer version.*

Description

Handles DiffusionMap, Sigmas, and GeneRelevance.

Usage

```
## S4 method for signature 'DiffusionMap'
updateObject(object, ..., verbose = FALSE)

## S4 method for signature 'Sigmas'
updateObject(object, ..., verbose = FALSE)

## S4 method for signature 'GeneRelevance'
updateObject(object, ..., verbose = FALSE)
```

Arguments

`object`	An object created with an older destiny release
`...`	ignored
`verbose`	tells what is being updated

Value

A DiffusionMap or Sigmas object that is valid when used with the current destiny release

Index