Package 'condiments'

October 18, 2022

Type Package

Title Differential Topology, Progression and Differentiation

Version 1.4.0

Description This package encapsulate many functions to conduct a differential topology analysis. It focuses on analyzing an 'omic dataset with multiple conditions. While the package is mostly geared toward scRNASeq, it does not place any restriction on the actual input format.

License MIT + file LICENSE

Encoding UTF-8

URL https://hectorrdb.github.io/condiments/index.html

Depends R (>= 4.0)

VignetteBuilder knitr

biocViews RNASeq, Sequencing, Software, SingleCell, Transcriptomics, MultipleComparison, Visualization

BugReports https://github.com/HectorRDB/condiments/issues

Imports slingshot (>= 1.9), mgcv, RANN, stats, SingleCellExperiment, SummarizedExperiment, utils, magrittr, dplyr (>= 1.0), Ecume (>= 0.9.1), methods, pbapply, matrixStats, BiocParallel, TrajectoryUtils, igraph, distinct

RoxygenNote 7.1.2

Suggests knitr, testthat, rmarkdown, covr, viridis, ggplot2, RColorBrewer, randomForest, tidyr, TSCAN

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

git_branch RELEASE_3_15

git_last_commit b0cab71

git_last_commit_date 2022-04-26

Date/Publication 2022-10-18

Author Hector Roux de Bezieux [aut, cre] (<https://orcid.org/0000-0002-1489-8339>), Koen Van den Berge [aut, ctb], Kelly Street [aut, ctb]

Maintainer Hector Roux de Bezieux <hector.rouxdebezieux@berkeley.edu>

R topics documented:

`create_differential_topology` *Create Example function*

Description

This creates a simulated reduced dimension dataset

Usage

```
create_differential_topology(
  n_cells = 200,
  noise = 0.15,
  shift = 10,
  unbalance_level = 0.9,
  speed = 1
)
```

Arguments

`n_cells`	The number of cells in the dataset.
`noise`	Amount of noise. Between 0 and 1.
`shift`	How much should the top lineage shift in condition B.
`unbalance_level`	How much should the bottom lineage be unbalanced toward condition A.
`speed`	How fast the cells from condition B should differentiate

Value

A list with two components

- sd: An n_cells by 4 dataframe that contains the reduced dimensions coordinates, lineage assignment (1 or 2) and condition assignment (A or B) for each cell.
- mst: a data.frame that contains the skeleton of the trajectories

Examples

```
sd <- create_differential_topology()
```

`differentiationTest` *Differential differentiation*

Description

Test whether or not the cell repartition between lineages is independent of the conditions

Usage

```
differentiationTest(...)
```

Arguments

...	See the fateSelectionTest

Value

See the fateSelectionTest

Examples

```
data('slingshotExample', package = "slingshot")
rd <- slingshotExample$rd
cl <- slingshotExample$cl
condition <- factor(rep(c('A','B'), length.out = nrow(rd)))
condition[110:139] <- 'A'
sds <- slingshot::slingshot(rd, cl)
differentiationTest(sds, condition)
```

`fateSelectionTest` *Differential fate selection Test*

Description

Test whether or not the cell repartition between lineages is independent of the conditions

Usage

```
fateSelectionTest(cellWeights, ...)

## S4 method for signature 'matrix'
fateSelectionTest(
  cellWeights,
  conditions,
  global = TRUE,
  pairwise = FALSE,
  method = c("Classifier", "mmd", "wasserstein_permutation"),
  classifier_method = "rf",
  thresh = 0.01,
  args_classifier = list(),
  args_mmd = list(),
  args_wass = list()
)

## S4 method for signature 'SlingshotDataSet'
fateSelectionTest(
  cellWeights,
  conditions,
  global = TRUE,
  pairwise = FALSE,
  method = c("Classifier", "mmd", "wasserstein_permutation"),
  classifier_method = "rf",
  thresh = 0.01,
  args_classifier = list(),
  args_mmd = list(),
  args_wass = list()
)

## S4 method for signature 'SingleCellExperiment'
fateSelectionTest(
  cellWeights,
  conditions,
  global = TRUE,
  pairwise = FALSE,
  method = c("Classifier", "mmd", "wasserstein_permutation"),
  classifier_method = "rf",
```

```
  thresh = 0.01,
  args_classifier = list(),
  args_mmd = list(),
  args_wass = list()
)

## S4 method for signature 'PseudotimeOrdering'
fateSelectionTest(
  cellWeights,
  conditions,
  global = TRUE,
  pairwise = FALSE,
  method = c("Classifier", "mmd", "wasserstein_permutation"),
  classifier_method = "rf",
  thresh = 0.01,
  args_classifier = list(),
  args_mmd = list(),
  args_wass = list()
)
```

Arguments

cellWeights	Can be either a SlingshotDataSet, a SingleCellExperiment object or a matrix of cell weights defining the probability that a cell belongs to a particular lineage. Each row represents a cell and each column represents a lineage. If only a single lineage, provide a matrix with one column containing all values of 1.
...	parameters including:
conditions	Either the vector of conditions, or a character indicating which column of the metadata contains this vector
global	If TRUE, test for all pairs simultaneously.
pairwise	If TRUE, test for all pairs independently.
method	One of "Classifier" or "mmd".
classifier_method	The method used in the classifier test. Default to "rf", i.e random forest.
thresh	The threshold for the classifier test. See details. Default to .05.
args_classifier	arguments passed to the classifier test. See classifier_test.
args_mmd	arguments passed to the mmd test. See mmd_test.
args_wass	arguments passed to the wasserstein permutation test. See wasserstein_permut.

Value

A data frame with 3 columns:

- *pair* for individual pairs, the lineages numbers. For global, "All".
- *p.value* the pvalue for the test at the global or pair level
- *statistic* The classifier accuracy

Examples

```
data('slingshotExample', package = "slingshot")
rd <- slingshotExample$rd
cl <- slingshotExample$cl
condition <- factor(rep(c('A','B'), length.out = nrow(rd)))
condition[110:139] <- 'A'
sds <- slingshot::slingshot(rd, cl)
fateSelectionTest(sds, condition)
```

`fateSelectionTest_multipleSamples` *Differential fate selection Test with multiple samples*

Description

Test whether or not the cell repartition between lineages is independent of the conditions, with samples not being confounded by conditions

Usage

```
fateSelectionTest_multipleSamples(cellWeights, ...)

## S4 method for signature 'matrix'
fateSelectionTest_multipleSamples(cellWeights, conditions, Samples, ...)

## S4 method for signature 'SlingshotDataSet'
fateSelectionTest_multipleSamples(cellWeights, conditions, Samples, ...)

## S4 method for signature 'SingleCellExperiment'
fateSelectionTest_multipleSamples(cellWeights, conditions, Samples, ...)

## S4 method for signature 'PseudotimeOrdering'
fateSelectionTest_multipleSamples(cellWeights, conditions, Samples, ...)
```

Arguments

`cellWeights`	Can be either a `SlingshotDataSet`, a `SingleCellExperiment` object or a matrix of cell weights defining the probability that a cell belongs to a particular lineage. Each row represents a cell and each column represents a lineage. If only a single lineage, provide a matrix with one column containing all values of 1.
`...`	Other arguments passed to `fateSelectionTest`.
`conditions`	Either the vector of conditions, or a character indicating which column of the metadata contains this vector.
`Samples`	A vector assigning each cell to a sample. Samples must be shared across all conditions.

Value

The same object has the fateSelectionTest with one more column per sample.

Examples

```
data('slingshotExample', package = "slingshot")
rd <- slingshotExample$rd
cl <- slingshotExample$cl
condition <- factor(rep(c('A','B'), length.out = nrow(rd)))
condition[110:139] <- 'A'
sds <- slingshot::slingshot(rd, cl)
samples <- sample(1:2, 140, replace = TRUE)
fateSelectionTest_multipleSamples(cellWeights = sds, conditions = condition, Samples = samples)
```

---

imbalance_score *Imbalance Score*

Description

Compute a imbalance score to show whether nearby cells have the same condition of not

Usage

```
imbalance_score(Object, ...)

## S4 method for signature 'matrix'
imbalance_score(Object, conditions, k = 10, smooth = 10)

## S4 method for signature 'SingleCellExperiment'
imbalance_score(Object, dimred = 1, conditions, k = 10, smooth = 10)
```

Arguments

Object	A SingleCellExperiment object or a matrix representing the reduced dimension matrix of the cells.
...	parameters including:
conditions	Either the vector of conditions, or a character indicating which column of the metadata contains this vector
k	The number of neighbors to consider when computing the score. Default to 10.
smooth	The smoothing parameter. Default to k. Lower values mean that we smooth more.
dimred	A string or integer scalar indicating the reduced dimension result in reducedDims(sce) to plot. Default to 1.

Value

Either a list with the `scaled_scores` and the `scores` for each cell, if input is a matrix, or the `SingleCellExperiment` object, wit this list in the `colData`.

Examples

```
data("toy_dataset")
scores <- imbalance_score(as.matrix(toy_dataset$sd[,1:2]),
  toy_dataset$sd$conditions, k = 4)
cols <- as.numeric(cut(scores$scaled_scores, 8))
plot(as.matrix(toy_dataset$sd[, 1:2]), xlab = "Dim1", ylab = "Dim2",
 pch = 16, col = RColorBrewer::brewer.pal(8, "Blues")[cols])
```

---

merge_sds *Merge slingshots datasets*

Description

If trajectory inference needs to be manually done condition per condition, this allows to merge them into one. It requires manual mapping of lineages.

Usage

```
merge_sds(..., mapping, condition_id = seq_len(ncol(mapping)), scale = FALSE)
```

Arguments

`...`	Slingshot datasets
`mapping`	a matrix, one column per dataset. Each row amounts to lineage mapping.
`condition_id`	A vector of condition for each condition. Default to integer values in order of appearance
`scale`	If TRUE (default), lineages that are mapped are scaled to have the same length.

Details

The function assumes that each lineage in a dataset maps to exactly one lineage in another dataset. Anything else needs to be done manually.

Value

A modified slingshot dataset that can be used for downstream steps.

Examples

```
data(list = 'slingshotExample', package = "slingshot")
if (!"cl" %in% ls()) {
  rd <- slingshotExample$rd
  cl <- slingshotExample$cl
}
sds <- slingshot::slingshot(rd, cl)
merge_sds(sds, sds, mapping = matrix(c(1, 2, 1, 2), nrow = 2))
```

`nLineages` *Number of lineages*

Description

Return the number of lineages for a slingshot object

Usage

```
nLineages(sds, ...)

## S4 method for signature 'SingleCellExperiment'
nLineages(sds)

## S4 method for signature 'SlingshotDataSet'
nLineages(sds)

## S4 method for signature 'PseudotimeOrdering'
nLineages(sds)
```

Arguments

`sds`	A slingshot object already run on the full dataset. Can be either a `SlingshotDataSet` or a `SingleCellExperiment` object.
`...`	parameters including:

Value

The number of lineages in the slingshot object

Examples

```
data(list = 'slingshotExample', package = "slingshot")
if (!"cl" %in% ls()) {
  rd <- slingshotExample$rd
  cl <- slingshotExample$cl
}
sds <- slingshot::slingshot(rd, cl)
nLineages(sds)
```

progressionTest *Differential Progression Test*

Description

Test whether or not the pseudotime distribution are identical within lineages between conditions

Usage

```
progressionTest(pseudotime, ...)

## S4 method for signature 'matrix'
progressionTest(
  pseudotime,
  cellWeights,
  conditions,
  global = TRUE,
  lineages = FALSE,
  method = ifelse(dplyr::n_distinct(conditions) == 2, "KS", "Classifier"),
  thresh = ifelse(method == "Classifer", 0.05, 0.01),
  args_mmd = list(),
  args_classifier = list(),
  args_wass = list(),
  rep = 10000,
  distinct_samples = NULL
)

## S4 method for signature 'SlingshotDataSet'
progressionTest(
  pseudotime,
  conditions,
  global = TRUE,
  lineages = FALSE,
  method = ifelse(dplyr::n_distinct(conditions) == 2, "KS", "Classifier"),
  thresh = ifelse(method == "Classifer", 0.05, 0.01),
  args_mmd = list(),
  args_classifier = list(),
  args_wass = list(),
  rep = 10000,
  distinct_samples = NULL
)

## S4 method for signature 'SingleCellExperiment'
progressionTest(
  pseudotime,
  conditions,
  global = TRUE,
```

```
  lineages = FALSE,
  method = ifelse(dplyr::n_distinct(conditions) == 2, "KS", "Classifier"),
  thresh = ifelse(method == "Classifer", 0.05, 0.01),
  args_mmd = list(),
  args_classifier = list(),
  args_wass = list(),
  rep = 10000,
  distinct_samples = NULL
)

## S4 method for signature 'PseudotimeOrdering'
progressionTest(
  pseudotime,
  conditions,
  global = TRUE,
  lineages = FALSE,
  method = ifelse(dplyr::n_distinct(conditions) == 2, "KS", "Classifier"),
  thresh = ifelse(method == "Classifer", 0.05, 0.01),
  args_mmd = list(),
  args_classifier = list(),
  args_wass = list(),
  rep = 10000,
  distinct_samples = NULL
)
```

Arguments

pseudotime	Can be either a SlingshotDataSet or a SingleCellExperiment object or a matrix of pseudotime values, each row represents a cell and each column represents a lineage.
...	parameters including:
cellWeights	If pseudotime is a matrix of pseudotime values, this represent the cell weights for each lineage. Ignored if pseudotime is not a matrix.
conditions	Either the vector of conditions, or a character indicating which column of the metadata contains this vector.
global	If TRUE, test for all lineages simultaneously.
lineages	If TRUE, test for all lineages independently.
method	One of "KS", "Classifier", "mmd", "wasserstein_permutation" or "Permutation" for a permutation. See details. Default to KS if there is two conditions and to "Classifier" otherwise.
thresh	The threshold for the KS test or Classifier test. Ignored if method = "Permutation". Default to .01 for KS and .05 for the 'classifier'.
args_mmd	arguments passed to the mmd test. See mmd_test.
args_classifier	arguments passed to the classifier test. See classifier_test.
args_wass	arguments passed to the wasserstein permutation test. See wasserstein_permut.

`rep` Number of permutations to run. Only for methods "Permutations" and "wasserstein_permutation". Default to `1e4`.

`distinct_samples`
The samples to which each cell belong to. Only use with method `distinct`. See `\code{\link{distinct_test}}` for help.

Details

For every lineage, we compare the pseudotimes of the cells from either conditions, using the lineage weights as observations weights.

- If `method = "KS"`, this uses the updated KS test, see `ks_test` for details.
- If `method = "Classifier"`, this uses a classifier to assess if that classifier can do better than chance on the conditions
- If `method = "Permutation"`, the difference of weighted mean pseudotime between condition is computed, and a p-value is found by permuting the condition labels.
- If `method = "mmd"`, this uses the mean maximum discrepancies statistics.

The p-value at the global level can be computed in two ways. method is `"KS"` or `"Permutation"`, then the p-values are computed using stouffer's z-score method, with the lineages weights acting as weights. Otherwise, the test works on multivariate data and is applied on all pseudotime values.

Value

A data frame with 3 columns:

- *lineage* for individual lineages, the lineage number. For global, `"All"`.
- *p.value* the pvalue for the test at the global or lineage level
- *statistic* for individual lineages, either the modified KS statistic if `method = "KS"`, or the weighted difference of means, if `method = "Permutation"`. For the global test, the combined Z-score.

References

Stouffer, S.A.; Suchman, E.A.; DeVinney, L.C.; Star, S.A.; Williams, R.M. Jr. (1949). *The American Soldier, Vol.1: Adjustment during Army Life.* Princeton University Press, Princeton.

Examples

```
data('slingshotExample', package = "slingshot")
rd <- slingshotExample$rd
cl <- slingshotExample$cl
condition <- factor(rep(c('A','B'), length.out = nrow(rd)))
condition[110:139] <- 'A'
sds <- slingshot::slingshot(rd, cl)
progressionTest(sds, condition)
```

progressionTest_multipleSamples
Differential Progression Test with multiple samples

Description

Test whether or not the pseudotime distribution are identical within lineages between conditions, with samples not being confounded by conditions

Usage

```
progressionTest_multipleSamples(pseudotime, ...)

## S4 method for signature 'matrix'
progressionTest_multipleSamples(
  pseudotime,
  cellWeights,
  conditions,
  Samples,
  ...
)

## S4 method for signature 'SlingshotDataSet'
progressionTest_multipleSamples(pseudotime, conditions, Samples, ...)

## S4 method for signature 'SingleCellExperiment'
progressionTest_multipleSamples(pseudotime, conditions, Samples, ...)

## S4 method for signature 'PseudotimeOrdering'
progressionTest_multipleSamples(pseudotime, conditions, Samples, ...)
```

Arguments

pseudotime	Can be either a SlingshotDataSet or a SingleCellExperiment object or a matrix of pseudotime values, each row represents a cell and each column represents a lineage.
...	Other arguments passed to progressionTest.
cellWeights	If ‘pseudotime‘ is a matrix of pseudotime values, this represent the cell weights for each lineage. Ignored if ‘pseudotime‘ is not a matrix.
conditions	Either the vector of conditions, or a character indicating which column of the metadata contains this vector.
Samples	A vector assigning each cell to a sample. Samples must be shared across all conditions.

Value

The same object has the progressionTest with one more column per sample.

Examples

```
data('slingshotExample', package = "slingshot")
rd <- slingshotExample$rd
cl <- slingshotExample$cl
condition <- factor(rep(c('A','B'), length.out = nrow(rd)))
condition[110:139] <- 'A'
sds <- slingshot::slingshot(rd, cl)
samples <- sample(1:2, 140, replace = TRUE)
progressionTest_multipleSamples(pseudotime = sds, conditions = condition, Samples = samples)
```

---

slingshot_conditions *Refitting slingshot per condition*

Description

Based on an original slingshot object, refit one trajectory per condition, using the same skeleton.

Usage

```
slingshot_conditions(sds, ...)

## S4 method for signature 'SlingshotDataSet'
slingshot_conditions(
  sds,
  conditions,
  approx_points = 100,
  adjust_skeleton = TRUE,
  verbose = TRUE,
  ...
)

## S4 method for signature 'SingleCellExperiment'
slingshot_conditions(
  sds,
  conditions,
  approx_points = 100,
  adjust_skeleton = TRUE,
  verbose = TRUE,
  ...
)

## S4 method for signature 'PseudotimeOrdering'
slingshot_conditions(
  sds,
  conditions,
  approx_points = 100,
  adjust_skeleton = TRUE,
```

```
  verbose = TRUE,
  ...
)
```

Arguments

sds	A slingshot object already run on the full dataset. Can be either a SlingshotDataSet or a SingleCellExperiment object.
...	Other arguments passed to getCurves
conditions	Either the vector of conditions, or a character indicating which column of the metadata contains this vector.
approx_points	Passed to getCurves
adjust_skeleton	Boolean, default to 'TRUE'. Whether to recompute the locations of the nodes after fitting per conditions.
verbose	Boolean, default to 'TRUE'. Control whether messages are printed.

Value

A list of SlingshotDataSet, one per condition.

Examples

```
data('slingshotExample', package = "slingshot")
rd <- slingshotExample$rd
cl <- slingshotExample$cl
condition <- factor(rep(c('A','B'), length.out = nrow(rd)))
condition[110:139] <- 'A'
sds <- slingshot::slingshot(rd, cl)
sdss <- slingshot_conditions(sds, condition)
```

---

topologyTest *Differential Topology Test*

Description

Test whether or not slingshot should be fitted independently for different conditions or not.

Usage

```
topologyTest(sds, ...)

## S4 method for signature 'SlingshotDataSet'
topologyTest(
  sds,
  conditions,
  rep = 100,
```

```
  threshs = 0.01,
  methods = ifelse(dplyr::n_distinct(conditions) == 2, "KS_mean", "Classifier"),
  parallel = FALSE,
  BPPARAM = BiocParallel::bpparam(),
  args_mmd = list(),
  args_classifier = list(),
  args_wass = list(),
  nmax = nrow(slingshot::slingPseudotime(sds)),
  distinct_samples = NULL
)

## S4 method for signature 'SingleCellExperiment'
topologyTest(
  sds,
  conditions,
  rep = 100,
  threshs = 0.01,
  methods = ifelse(dplyr::n_distinct(conditions) == 2, "KS_mean", "Classifier"),
  parallel = FALSE,
  BPPARAM = BiocParallel::bpparam(),
  args_mmd = list(),
  args_classifier = list(),
  args_wass = list(),
  nmax = ncol(sds),
  distinct_samples = NULL
)

## S4 method for signature 'PseudotimeOrdering'
topologyTest(
  sds,
  conditions,
  rep = 100,
  threshs = 0.01,
  methods = ifelse(dplyr::n_distinct(conditions) == 2, "KS_mean", "Classifier"),
  parallel = FALSE,
  BPPARAM = BiocParallel::bpparam(),
  args_mmd = list(),
  args_classifier = list(),
  args_wass = list(),
  nmax = nrow(slingshot::slingPseudotime(sds)),
  distinct_samples = NULL
)
```

Arguments

sds	A slingshot object already run on the full dataset. Can be either a SlingshotDataSet or a SingleCellExperiment object.
...	parameters including:

conditions	Either the vector of conditions, or a character indicating which column of the metadata contains this vector.
rep	How many permutations to run. Default to 50.
threshs	the threshold(s) for the KS test or classifier test. Default to .01 See ks_test and classifier_test.
methods	The method(s) to use to test. Must be among 'KS_mean', 'Classifier', "KS_all", "mmd' and 'wasserstein_permutation'. See details.
parallel	Logical, defaults to FALSE. Set to TRUE if you want to parallellize the fitting.
BPPARAM	object of class bpparamClass that specifies the back-end to be used for computations. See bpparam in BiocParallel package for details.
args_mmd	arguments passed to the mmd test. See mmd_test.
args_classifier	arguments passed to the classifier test. See classifier_test.
args_wass	arguments passed to the wasserstein permutation test. See wasserstein_permut.
nmax	How many samples to use to compute the mmd test. See details.
distinct_samples	The samples to which each cell belong to. Only use with method 'distinct'. See 'distinct_test' for help.

Details

If there is only two conditions, default to 'KS_mean'. Otherwise, uses a classifier.

More than one method can be specified at once, which avoids running slingshot on the permutations more than once (as it is the slowest part).

For the 'mmd_test', if 'null=unbiased', it is recommand to set 'nmax=2000' or something of that order of magnitude to avoid overflowing the memory.

Value

A list containing the following components:

- *method* The method used to test
- *thresh* The threshold (if relevant)
- *statistic* the value of the test statistic.
- *p.value* the p-value of the test.

Examples

```
data('slingshotExample', package = "slingshot")
rd <- slingshotExample$rd
cl <- slingshotExample$cl
condition <- factor(rep(c('A','B'), length.out = nrow(rd)))
condition[110:139] <- 'A'
sds <- slingshot::getLineages(rd, cl)
topologyTest(sds, condition, rep = 10)
```

topologyTest_multipleSamples
Differential Topology Test with multiple samples

Description

Test whether or not slingshot should be fitted independently for different conditions or not, per sample, with samples not being confounded by conditions.

Usage

```
topologyTest_multipleSamples(sds, ...)

## S4 method for signature 'SlingshotDataSet'
topologyTest_multipleSamples(sds, conditions, Samples, ...)

## S4 method for signature 'SingleCellExperiment'
topologyTest_multipleSamples(sds, conditions, Samples, ...)

## S4 method for signature 'PseudotimeOrdering'
topologyTest_multipleSamples(sds, conditions, Samples, ...)
```

Arguments

sds	A slingshot object already run on the full dataset. Can be either a SlingshotDataSet or a SingleCellExperiment object.
...	Other arguments passed to topologyTest.
conditions	Either the vector of conditions, or a character indicating which column of the metadata contains this vector.
Samples	A vector assigning each cell to a sample. Samples must be shared across all conditions.

Value

The same object has the topologyTest with one more column per sample.

Examples

```
data('slingshotExample', package = "slingshot")
rd <- slingshotExample$rd
cl <- slingshotExample$cl
condition <- factor(rep(c('A','B'), length.out = nrow(rd)))
condition[110:139] <- 'A'
sds <- slingshot::slingshot(rd, cl)
samples <- sample(1:2, 140, replace = TRUE)
topologyTest_multipleSamples(sds = sds, conditions = condition,
                             Samples = samples, rep = 10)
```

toy_dataset *A toy dataset used in the vignette and in the examples*

Description

This example has been created using the 'create_differential_topology' function.

Usage

```
data(toy_dataset)
```

Format

A list with two dataframes

- *sd* A dataframe containing, for 1000 cells, the dimensions in two coordinates, and cluster, lineage and condition assignment.
- `mst`: a data.frame that contains the skeleton of the trajectories

Source

The following code reproduces the object `set.seed(21) library(condiments) data <- create_differential_topolog = 1000, shift = 0) data$sd$Dim2 <- data$sd$Dim2 * 5 data$mst$Dim2 <- data$mst$Dim2 * 5 data$sd$cl <- kmeans(as.matrix(data$sd[, 1:2]), 8)$cluster data$sd$cl <- as.character(data$sd$cl)`

weights_from_pst *weights_from_pst*

Description

Most trajectory inference methods do not perform soft assignment but instead assign cells to all possible lineages before a branching point, and then to one or another. This function re-creates a weight matrix from those matrices of pseudotime

Usage

```
weights_from_pst(pseudotime, ...)

## S4 method for signature 'matrix'
weights_from_pst(pseudotime)

## S4 method for signature 'data.frame'
weights_from_pst(pseudotime)
```

Arguments

pseudotime	A matrix or data.frame of \[ncells\] by \[nCurves\].
...	Other parameters including:

Value

A object of the same type and dimensions as the original object, with the weights for each curve and cell.

Examples

```
data(list = 'slingshotExample', package = "slingshot")
if (!"cl" %in% ls()) {
  rd <- slingshotExample$rd
  cl <- slingshotExample$cl
}
sds <- slingshot::slingshot(rd, cl)
weights_from_pst(slingshot::slingPseudotime(sds))
```

Index