Package ‘CellTrails’

December 4, 2025

Type Package

Title Reconstruction, visualization and analysis of branching trajectories

Version 1.29.0

Description CellTrails is an unsupervised algorithm for the de novo chronological ordering, visualization and analysis of single-cell expression data. CellTrails makes use of a geometrically motivated concept of lower-dimensional manifold learning, which exhibits a multitude of virtues that counteract intrinsic noise of single cell data caused by drop-outs, technical variance, and redundancy of predictive variables. CellTrails enables the reconstruction of branching trajectories and provides an intuitive graphical representation of expression patterns along all branches simultaneously. It allows the user to define and infer the expression dynamics of individual and multiple pathways towards distinct phenotypes.

License Artistic-2.0

Encoding UTF-8

VignetteBuilder knitr

Depends R (>= 3.5), SingleCellExperiment

Imports BiocGenerics, Biobase, cba, dendextend, dtw, EnvStats, ggplot2, ggrepel, grDevices, igraph, maptree, methods, mgcv, reshape2, Rtsne, stats, splines, SummarizedExperiment, utils

Suggests AnnotationDbi, destiny, RUnit, scater, scran, knitr, org.Mm.eg.db, rmarkdown

RoxygenNote 7.1.0

biocViews ImmunoOncology, Clustering, DataRepresentation, DifferentialExpression, DimensionReduction, GeneExpression, Sequencing, SingleCell, Software, TimeCourse

Collate 'AllClasses.R' 'AllGenerics.R' 'accessor-methods.R' 'cluster-methods.R' 'data-sets.R' 'diffexp-methods.R' 'dimred-methods.R' 'export-methods.R' 'filter-methods.R' 'fitting-methods.R' 'import-methods.R' 'imputation-methods.R' 'layout-methods.R' 'plot-methods.R' 'show-methods.R' 'simulation-methods.R' 'test-methods.R' 'trajectory-methods.R' 'utility-methods.R'

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

git_branch devel

git_last_commit 75d815a

git_last_commit_date 2025-10-29

Repository Bioconductor 3.23

Date/Publication 2025-12-04

Author Daniel Ellwanger [aut, cre, cph]

Maintainer Daniel Ellwanger <`dc.ellwanger.dev@gmail.com`>

Contents

`.adjustLayoutByPtime` *Adjusting the trajectory graph layout*

Description

Adjusts the edge distances in the trajectory graph such that edge distances correlates to edge weights; pseudotime is stored in edge weights

Usage

```
.adjustLayoutByPtime(x, l)
```

Arguments

`x`	A `SingleCellExperiment` object
`layout`	Layout coordinates for each sample

Value

An adjusted layout

Author(s)

Daniel C. Ellwanger

`.bhtsne` *#' DEF: Truncate eigenbasis #' #' For details see* `reduceDimensions` *#' @keywords internal #' @author Daniel C. Ellwanger .reduceDimensions_def <- function(x, s) CellTrails::latentSpace(x) <- CellTrails::latentSpace(x)[, seq_len(s@n)] CellTrails::eigenvalues(x) <- CellTrails::eigenvalues(x)[seq_len(s@n)] x t-Distributed Stochastic Neighbor Embedding*

Description

Barnes-Hut implementation of t-Distributed Stochastic Neighbor Embedding

Usage

```
.bhtsne(x, dims = 2, perplexity = 30, theta = 0.5, max_iter = 1000)
```

Arguments

`x`	A numerical matrix
`dims`	Output dimensionality
`perplexity`	Perplexity parameter (default: 30)
`theta`	Speed/accuracy trade-off (increase for less accuracy), set to 0.0 for exact tSNE (default: .5)
`max_iter`	Number of iterations (default: 1000)

Value

A list with the following components:

`Y` Matrix containing the new representations for the objects

`perplexity` See above

Author(s)

Daniel C. Ellwanger

`.capitalize` *Capitalizes first character of string*

Description

Capitalizes first character of string and sets the rest to lower case.

Usage

```
.capitalize(x)
```

Arguments

`x`	A string

Details

Example: "abC" becoms "Abc".

Value

A string

Author(s)

Daniel C. Ellwanger

`.color_hue` *Color palette*

Description

Generates equally spaced hues around the color wheel

Usage

```
.color_hue(n)
```

Arguments

`n`	Number of colors to be generated

Value

Colors codes

Author(s)

Daniel C. Ellwanger

`.color_ramp` *Colors for vector*

Description

Generates colors for vector

Usage

```
.color_ramp(
  x,
  range = 3,
  colPal = NULL,
  min.val = 1e-10,
  min.val.col = NA,
  breaks = 25
)
```

Arguments

`x`	A numeric vector

Value

Color codes

Author(s)

Daniel C. Ellwanger

`.connectStates_def` *DEF: Connect states*

Description

For details see `connectStates`

Usage

```
.connectStates_def(dmat, cl, l = 10, sigma = 1)
```

Arguments

`dmat`	Distance matrix
`cl`	States

Author(s)

Daniel C. Ellwanger

`.connect_ordi` *AUX Generate graph based on ordination*

Description

Generates 2D ordination from orthogonal projection

Usage

```
.connect_ordi(ordi)
```

Arguments

`ordi`	Ordination

Value

An `igraph` object

Author(s)

Daniel C. Ellwanger

`.connect_ortho` *AUX: Connect orthogonal projected samples*

Description

Connects samples based on their projected position on the trajectory.

Usage

```
.connect_ortho(Xorth)
```

Arguments

`Xorth`	Result from the orthogonal projection

Value

An `igraph` object

Author(s)

Daniel C. Ellwanger

`.contrastExprTrail_def` *DEF: Differential expression between trails*

Description

DEF: Differential expression between trails

Usage

```
.contrastExprTrail_def(
  ptime_1,
  ptime_2,
  feature_expr,
  sts,
  trail_names,
  n = 250,
  score,
  dynamicFit_k = 5,
  df = 10
)
```

Arguments

`ptime_1`	Pseudotime trail 1
`ptime_2`	Pseudotime trail 2
`feature_expr`	Expression vector
`sts`	Samples vector
`n`	Number of fitted values
`score`	Score type
`dynamicFit_k`	Freedom param for dynamic fit
`df`	Freedom param for splines fit

Author(s)

Daniel C. Ellwanger

`.deleteMedianCentres` *AUX Remove median centres*

Description

Deletes median centres from trajectory graph if graph has not been simplified/has been expanded.

Usage

```
.deleteMedianCentres(g)
```

Arguments

`g`	An object of class `igraph`

Value

An unpdated object of class `igraph`

Author(s)

Daniel C. Ellwanger

`.denoiseExpression` *Denoises expression*

Description

Blocks factors in expression matrix

Usage

```
.denoiseExpression(x, design)
```

Arguments

`x`	A numeric expression vector
`design`	A numeric matrix describing the factors that should be blocked

Value

A numeric denoised expression vector

Author(s)

Daniel C. Ellwanger

`.diffExpr` *Compute differential expression*

Description

Computes P-value and fold-change between two expression vectors.

Usage

```
.diffExpr(x, y, lod = NULL, alternative = "two.sided")
```

Arguments

`x`	Feature expression in condition x
`y`	Feature expression in condition y
`lod`	Gene limit of detection

Details

For censored data a Peto-Peto test is performed, for non-censored data a Wilcoxon rank sum test. If limit of detection is provided, expectation maximization is used to compute the fold-change.

Value

A list containing the following components:

`p.value`	P-value
`fold`	Fold-change

Author(s)

Daniel C. Ellwanger

`.diffExprState_def` *DEF: Differential expression between states*

Description

DEF: Differential expression between states

Usage

```
.diffExprState_def(
  x,
  state1,
  state2,
  feature_name,
  alternative = "two.sided",
  lod = NULL
)
```

Author(s)

Daniel C. Ellwanger

`.embedSamples_def` *DEF: Spectral embedding of samples*

Description

For details see `embedSamples`

Usage

```
.embedSamples_def(x, nbins = 10, sigma = 0.75, design = NULL)
```

Arguments

`x`	A numerical matrix
`nbins`	Cubic B-spline discretization is used to compute fuzzy mutual information between pairs of samples; `nbins` defines the number of intervals used for discretization of expression data. (default: 10)
`sigma`	The mutual information matrix is transformed to an adjacency matrix using a heat kernel; `sigma` defines the radius of heat kernel (in quantiles; default: `sigma` = 0.75 which is the third quantile of the mutual information matrix).
`design`	A numeric matrix describing the factors that should be blocked

Author(s)

Daniel C. Ellwanger

`.exprs` *GET expression matrix*

Description

Retrieve numeric matrix of expression values for processing in CellTrails. This wrapper function ensures that all functions in the package receive the proper assay from the `SingleCellExperiment` object.

Usage

```
.exprs(object)
```

Arguments

`object`	An object of class `SingleCellExperiment`

Value

A `numeric` matrix

Author(s)

Daniel C. Ellwanger

`.featureNameExists` *Checks if feature exists*

Description

Checks if feature exists

Usage

```
.featureNameExists(x, feature_name)
```

Arguments

`x`	A `SingleCellExperiment` object
`feature_name`	Name of feature

Value

logical value

Author(s)

Daniel C. Ellwanger

`.filterTrajFeaturesByCOV_def` *DEF: Filter features by coefficient of variation*

Description

For details see `filterFeaturesByCOV`

Usage

```
.filterTrajFeaturesByCOV_def(y, threshold, design = NULL, show_plot = TRUE)
```

Arguments

`y`	An expression vector
`threshold`	numeric cutoff value
`design`	Model matrix
`show_plot`	Show plot?

Author(s)

Daniel C. Ellwanger

`.filterTrajFeaturesByDL_def` *DEF: Filter feaures by Detection Level*

Description

For details see `filterFeaturesByPOD`

Usage

```
.filterTrajFeaturesByDL_def(y, threshold, show_plot = TRUE)
```

Arguments

`y`	An expression matrix
`threshold`	numeric cutoff value
`show_plot`	Show plot?

Author(s)

Daniel C. Ellwanger

`.filterTrajFeaturesByFF_def` *DEF: Filter features by index of dispersion / fano factor*

Description

For details see `filterFeaturesByFF`

Usage

```
.filterTrajFeaturesByFF_def(
  y,
  z,
  min_expr = 0,
  design = NULL,
  show_plot = TRUE
)
```

Arguments

`y`	An expression vector
`z`	A cutoff z-score
`min_expr`	Minimum expression level
`design`	Model matrix
`show_plot`	Show plot?

Author(s)

Daniel C. Ellwanger

`.findSpectrum_def` *DEF: Determine number of informative latent dimensions*

Description

For details see `findSpectrum`

Usage

```
.findSpectrum_def(D, frac = 100)
```

Author(s)

Daniel C. Ellwanger

`.findStates_def` *DEF: Find states*

Description

For details see `findStates`

Usage

```
.findStates_def(
  X,
  ordi,
  link.method = "ward.D2",
  min.size,
  max.pval = 1e-04,
  min.fc = 2,
  min.g = 5,
  reverse = FALSE,
  verbose = FALSE
)
```

Arguments

`X`	Expression matrix
`ordi`	Ordination of samples
`link.method`	Linkage criteria
`min.size`	Min size of inital clusters
`max.pval`	Pval threshold
`min.fc`	Fold-change threshold
`reverse`	Reverse order
`verbose`	For debug
`ming.g`	Feature cound threshold

Author(s)

Daniel C. Ellwanger

`.fitDynamic_def` *Fitting dynamics*

Description

Fits expression as a function of pseudotime using generalized additive models

Usage

```
.fitDynamic_def(x, y, z, k = 5, n.out = NULL, x.out = NULL)
```

Arguments

`x`	Pseudotime values
`y`	Expression values
`z`	Sample states
`k`	Dimensions of the bases used to represent the smooth term
`n.out`	Number of predicted expression values within pseudotime range
`x.out`	Predicted expression values for given set of pseudotime values

Value

A list containing the following components:

`x`	Pseudotime
`y`	Fitted values
`mod`	GAM

Author(s)

Daniel C. Ellwanger

`.fitTrajectory_def` *DEF: Trajectory fitting*

Description

For details see `fitTrajectory`

Usage

```
.fitTrajectory_def(cl = cl, g = g, X = X, snames)
```

Arguments

`cl`	Trajectory states vector
`g`	Trajectory graph; `igraph` object
`X`	Lower-dimensional ordination of trajectory samples
`snames`	Sample names

Author(s)

Daniel C. Ellwanger

`.fit_surface` *Fitting the map surface*

Description

Fits the smooth surface of CellTrails maps

Usage

```
.fit_surface(X, y, npoints = 300, weights = NULL, knots = 10, rescale = FALSE)
```

Arguments

`X`	Ordination; numerical matrix
`y`	Expression values; numerical vector
`npoints`	Number of points along x and y axis
`weights`	Cluster states (defines weights based on fraction of non-detects per state)
`knots`	Number of knots in spline function

Value

A list containing the following components:

`fit`	GAM fit object
`grid`	Fitted values

Author(s)

Daniel C. Ellwanger

`.fr_layout` *Computes state trajectory graph layout*

Description

Uses the Fruchterman-Reingold layout algorithm

Usage

```
.fr_layout(g)
```

Arguments

`g`	An igraph graph object

Value

numerical matrix with the layout coordinates

Author(s)

Daniel C. Ellwanger

`.generate_ordination` *AUX: Orthogonal projection ordination*

Description

Generates 2D ordination from orthogonal projection

Usage

```
.generate_ordination(g, error, factor = 7, rev = FALSE, only.ordi = FALSE)
```

Arguments

`g`	Graph from orthogonal projection, `igraph` object
`error`	Error from orthogonal projection; `numeric` vector
`only.ordi`	Compute only ordination?

Value

A list containing the following components:

`ordi`	Ordination
`ordi.jitter`	Jittered ordination
`backbone`	Backbone of trajectory

Author(s)

Daniel C. Ellwanger

`.ihs` *Reverse inverse hyperbolic sine*

Description

Computes inverse hyperbolic sine

Usage

```
.ihs(x)
```

Arguments

`x`	A numeric value, vector or matrix

Details

In contrast to sqrt and log, ihs is defined for negative numbers.

Value

Transformed values

Author(s)

Daniel C. Ellwanger

`.linear_fit` *Linear fit*

Description

Perform simple linear fit

Usage

```
.linear_fit(x.in, y.in, x.out = NULL)
```

Arguments

`x.in`	Predictor variable, numeric vector
`y.in`	Response variable, numeric vector
`x.out`	Predictor variable for linear function, numeric vector

Value

A list containing the following components:

`p.value`	Anova P-value
`r2`	Adjusted R-squared
`x.out`	Predictor variable for linear function, numeric vector
`y.out`	Response from linear function

Author(s)

Daniel C. Ellwanger

`.needsToBeExpanded` *AUX Check if projection needs to be expanded*

Description

Simple graphs having only bifurcations along the backbone get simplified by a 2D projection. If a fork with more than two successors or a fork not located along the backbone, but along a side branch, the 2D simplification would collate the states and ignore the bifurcation. Therefore, the simplification step has to be skipped. This methods checks whether a siplification is possible or not.

Usage

```
.needsToBeExpanded(g, cl)
```

Arguments

`g`	Trajectory graph; `igraph` object
`cl`	Trajectory states vector; logical

Details

IF: any node has a degree > 3, return `TRUE`; OTHERWISE: Check all shortest paths from start node to any leaf in trajectory tree. IF any path passes a node with degree > 3, which is not located on the backbone, return `TRUE`; OTHERWISE: return `FALSE`.

Value

A logical value

Author(s)

Daniel C. Ellwanger

`.nn_impute` *Nearest neighbor imputation*

Description

Nearest neighbor imputation

Usage

```
.nn_impute(y, D)
```

Arguments

`y`	Values; missing values need to be set to NA
`D`	distance matrix between samples

Value

Imputed values

Author(s)

Daniel C. Ellwanger

.pca_def *DEF: PCA*

Description

For details see pca

Usage

```
.pca_def(M, do_scaling = TRUE, design = NULL)
```

Arguments

M	expression matrix
do_scaling	use covariance/correlation matrix
design	Block factors

Author(s)

Daniel C. Ellwanger

.pheno *GET phenotype values*

Description

Returns phenotype values from SingleCellExperiment object

Usage

```
.pheno(object, name)
```

Arguments

object	A SingleCellExperiment object
name	Name of phenotype

Details

Wrapper for colDat(object)[, name] which also accesses internal metadata (e.g., landmarks).

Value

A vector of any type

Author(s)

Daniel C. Ellwanger

`.phenoNameExists` *Checks if phenotype exists*

Description

Checks if phenotype exists

Usage

```
.phenoNameExists(x, pheno_name)
```

Arguments

`x`	A `SingleCellExperiment` object
`pheno_name`	Name of phenotype

Value

logical value

Author(s)

Daniel C. Ellwanger

`.plotDynamic` *DEF: Visualize expression dynamic*

Description

Shows feature expression as a function of pseudotime

Usage

```
.plotDynamic(x, Y, feature_name, trail_name, weights, k = 5)
```

Arguments

`x`	Pseudotime
`Y`	Expression matrix (samples in columns)
`weights`	Weight (states)
`k`	Dimensions of bases used to represent the smooth term

Author(s)

Daniel C. Ellwanger

.plotManifold_def *DEF: Visualizing the manifold*

Description

Plots approximation of sample embedding in latent space in two dimensions

Usage

```
.plotManifold_def(
  X,
  y,
  name,
  g = NULL,
  weights = NULL,
  axis_label = "",
  colors = c("pretty", "bw"),
  type = c("raw", "surface.fit", "surface.se"),
  samples_only = FALSE,
  setND = FALSE,
  alpha = 0.1
)
```

Arguments

X	Ordination of data points
y	Values to visualize
name	Label for the figure legend
g	The trajectory graph
weights	States along the trajectory
axis_label	Label prefix for each axis
colors	Either the color ramp is colorized or black/white
type	Type of visualization
samples_only	Does not show the whole surface, but colorizes only the samples
setND	Indicates if a '0' value should be set to 'ND' in the figure legend
alpha	Color param

Value

A ggplot object

Author(s)

Daniel C. Ellwanger

`.plotSpectrum_def` *DEF: Visualizing the spectrum definition*

Description

Plots Scree plot with eigengaps

Usage

```
.plotSpectrum_def(x)
```

Arguments

`x`	A `list`

Value

A `ggplot` object

Author(s)

Daniel C. Ellwanger

`.plotStateExpression_def` *DEF: Violine plots of genes*

Description

Plot gene expression per state

Usage

```
.plotStateExpression_def(x, sts, label)
```

Arguments

`x`	A expression vector
`sts`	A states assignment vector
`label`	Label for legend

Value

A `ggplot` object

Author(s)

Daniel C. Ellwanger

`.plotStateSize_def` *DEF: Visualizing states sizes*

Description

Plots barplot of number of samples per state

Usage

```
.plotStateSize_def(x)
```

Arguments

`x`	States

Value

A `ggplot` object

Author(s)

Daniel C. Ellwanger

`.plotStateTrajectory_def` *DEF: Visualizing the trajectory graph*

Description

Visualizes the trajectory spanning all states of a component

Usage

```
.plotStateTrajectory_def(
  X,
  g,
  y,
  all_sts,
  name,
  setND = FALSE,
  point_size = 3,
  label_offset = 2
)
```

Arguments

`X`	Layout
`g`	State trajectory graph (igraph object)
`y`	Values; numeric or factor vector
`all_sts`	All state assignments
`setND`	If values of '0' are getting set to 'ND' in the figure legend
`point_size`	Point size paramenter
`label_offset`	Label offset parameter
`label`	Label for plot legend

Value

A `ggplot` object

Author(s)

Daniel C. Ellwanger

`.plotTrailblazing_def` *DEF: Visualizing trailblazing*

Description

Shows trajectory graph and highlights landmarks

Usage

```
.plotTrailblazing_def(X, g, ltype, lid)
```

Arguments

`X`	The trajectory layout
`g`	The trajectory graph
`ltype`	Vector with landmark types
`lid`	Vector with landmark ids

Value

A `ggplot` object

Author(s)

Daniel C. Ellwanger

`.plotTrail_def` *DEF: Visualize trail on map*

Description

Highlights a trail on map

Usage

```
.plotTrail_def(X, g, ptime, name)
```

Arguments

`X`	The trajectory layout
`g`	The trajectory graph
`ptime`	The trail pseudotime
`name`	Name of trail

Author(s)

Daniel C. Ellwanger

`.plot_trajectoryFit` *DEF: Visualizing the trajectory fit*

Description

Shows the residuals along the trajectory backbone

Usage

```
.plot_trajectoryFit(x, g, sts, factor = 7, rev = FALSE)
```

Arguments

`x`	The fitting residuals
`g`	The trajectory graph
`sts`	The states along the trajectory
`factor`	The jitter intensity correlates to the projection error
`rev`	Should the trajectory shown in reverse order?

Value

A `ggplot` object

Author(s)

Daniel C. Ellwanger

`.prettyColorRamp` *Pretty color ramp*

Description

Pretty color ramp

Usage

```
.prettyColorRamp(n, grayStart = TRUE)
```

Arguments

`n`	Number of colors

Value

Color codes

Author(s)

Daniel C. Ellwanger

`.prettyString` *Pretty string from array*

Description

Generates short representation of long character vectors

Usage

```
.prettyString(x, mmax = 4)
```

Arguments

`x`	A character vector

Details

Returns pretty print of character vector

Value

String

Author(s)

Daniel C. Ellwanger

`.project_ortho` *AUX: Orthogonal projection*

Description

Orthogonally projects samples onto trajectory

Usage

```
.project_ortho(cl, g, X)
```

Arguments

`cl`	Trajectory states; factor vector
`g`	Trajectory tree; igraph object
`X`	Lower dimensional ordination of trajectory samples

Value

A `list` containing the following components:

`Y`	Projection
`X.c`	Mediancentres
`lambda`	Projection lambda (position: inside/outside of edge)
`adjmat`	Adjacency matrix with sample count per edge
`error`	Projection error per sample
`edgeId`	ID of each edge
`edgeId2Edge`	Mapping of edge to edgeId
`edge`	For each sample its assigned edge

Author(s)

Daniel C. Ellwanger

.rbf *Radial basis function*

Description

Computes radial basis function

Usage

```
.rbf(d, radius)
```

Arguments

x	A numeric value, vector or matrix
sigma	Radius of the kernel

Details

Also known as Heat kernel or Gaussian kernel

Value

Transformed values

Author(s)

Daniel C. Ellwanger

.rescale *Rescale vector*

Description

Rescales vector to [ymin, ymax]

Usage

```
.rescale(x, ymin, ymax)
```

Arguments

x	X-values
ymin	New min
ymax	New max

Value

Rescaled vector

Author(s)

Daniel C. Ellwanger

`.sampleNameExists` *Checks if sample exists*

Description

Checks if sample exists

Usage

```
.sampleNameExists(x, sample_name)
```

Arguments

`x`	A `SingleCellExperiment` object
`sample_name`	Name of sample

Value

logical value

Author(s)

Daniel C. Ellwanger

`.spanForest` *GET state trajectory graph*

Description

Returns graph object spanning all states (spanning forest)

Usage

```
.spanForest(object)
```

Arguments

`object`	A `SingleCellExperiment` object

Value

A `list` object with an `igraph` object per component of the spanning forest

Author(s)

Daniel C. Ellwanger

`.spanForest<-` *SET state trajectory graph*

Description

Sets graph object spanning all states (spanning forest) to `SingleCellExperiment` object

Usage

```
.spanForest(object) <- value
```

Arguments

`object`	A `SingleCellExperiment` object
`value`	A `list` object with an `igraph` object per component of the spanning forest

Value

An updated object of class `SingleCellExperiment`

Author(s)

Daniel C. Ellwanger

`.spatmed` *Spatial median*

Description

Computes mediancentres

Usage

```
.spatmed(X, maxiter = 500, eps = 1e-06)
```

Arguments

`x`	A numeric matrix

Value

A numeric vector

Author(s)

Daniel C. Ellwanger

`.stateTrajLayout` *GET state trajectory layout*

Description

Gets layout of state trajectory from `SingleCellExperiment` object

Usage

```
.stateTrajLayout(object, component)
```

Arguments

`object`	A `SingleCellExperiment` object

Value

A `numeric` matrix

Author(s)

Daniel C. Ellwanger

`.trailNameExists` *Checks if trail exists*

Description

Checks if trail exists

Usage

```
.trailNameExists(x, trail_name)
```

Arguments

`x`	A `SingleCellExperiment` object
`trail_name`	Name of trail

Value

logical value

Author(s)

Daniel C. Ellwanger

`.trajGraph` *GET trajectory graph*

Description

Returns trajectory graph from a `SingleCellExperiment` object

Usage

```
.trajGraph(object)
```

Arguments

`object`	A `SingleCellExperiment` object

Value

A `igraph` object

Author(s)

Daniel C. Ellwanger

`.trajGraph<-` *SET trajectory graph*

Description

Stores trajectory graph in a `SingleCellExperiment` object

Usage

```
.trajGraph(object) <- value
```

Arguments

`object`	A `SingleCellExperiment` object
`value`	A `igraph` object

Value

An updated object of class `SingleCellExperiment`

Author(s)

Daniel C. Ellwanger

`.trajLandmark` *GET trajectory landmark annotation*

Description

Returns trajectory landmark information from `SingleCellExperiment` object

Usage

```
.trajLandmark(object, type = c("type", "id", "shape"))
```

Arguments

`object`	A `SingleCellExperiment` object
`type`	A character in "type", "id", "shape"

Value

A vector of any type

Author(s)

Daniel C. Ellwanger

`.trajLandmark<-` *SET trajectory landmark annotation*

Description

Stores information on trajectory landmarks in a `SingleCellExperiment` object

Usage

```
.trajLandmark(object, type = c("type", "id", "shape")) <- value
```

Arguments

`object`	A `SingleCellExperiment` object
`type`	A character in "type", "id", "shape"
`value`	A vector of any type

Value

An updated object of class `SingleCellExperiment`

Author(s)

Daniel C. Ellwanger

`.trajResiduals<-` *SET trajectory fitting residuals*

Description

Stores trajectory fitting residuals in `SingleCellExperiment` object

Usage

```
.trajResiduals(object) <- value
```

Arguments

`object`	A `SingleCellExperiment` object
`value`	A `numeric` vector

Value

An updated object of class `SingleCellExperiment`

Author(s)

Daniel C. Ellwanger

`.useFeature` *GET trajectory features indicator*

Description

Indicates if feature should be used for trajectory reconstruction. Spike-in controls are removed.

Usage

```
.useFeature(object)
```

Arguments

`object`	An object of class `SingleCellExperiment`

Value

A `logical` vector

Author(s)

Daniel C. Ellwanger

`.useFeature<-` *SET trajectory features indicator*

Description

Sets indicator if feature should be used for trajectory reconstruction.

Usage

```
.useFeature(object) <- value
```

Arguments

`object`	An object of class `SingleCellExperiment`
`value`	A logical vector

Value

An updated object of class `SingleCellExperiment`

Author(s)

Daniel C. Ellwanger

`.useSample` *GET trajectory samples indicator*

Description

Indicates if sample was used for trajectory reconstruction.

Usage

```
.useSample(object)
```

Arguments

`object`	An object of class `SingleCellExperiment`

Value

A `logical` vector

Author(s)

Daniel C. Ellwanger

`.useSample<-` *SET trajectory samples indicator*

Description

Sets indicator if sample was used for trajectory reconstruction.

Usage

```
.useSample(object) <- value
```

Arguments

`object`	An object of class `SingleCellExperiment`
`value`	A logical vector

Value

An updated object of class `SingleCellExperiment`

Author(s)

Daniel C. Ellwanger

`.validatePlotParams` *Validates parameter settings for plot methods*

Description

Validates parameter settings for plot methods

Usage

```
.validatePlotParams(x, color_by, name)
```

Arguments

`x`	A `SingleCellExperiment` object
`color_by`	Color_by parameter
`name`	Name parameter

Details

Parameter `color_by` needs to be in 'featureName', 'phenoName'. Parameter `name` needs to be existent in rownames(object) or in colnames(colData(object)), state, landmark. If parameters were validated true, this function automatically extracts and returns the values. Character vectors get converted to factors.

Value

A `list` with the parameters and values

Author(s)

Daniel C. Ellwanger

`.write_ygraphml_def` *DEF: Export trajectory graph*

Description

For details see `write.ygraphml`

Usage

```
.write_ygraphml_def(g, X, file, col_values, lbl_values, shapes)
```

Arguments

`g`	Trajectory graph
`X`	Lower-dimensional ordination of trajectory samples
`col_values`	Color values
`lbl_values`	Label values
`shapes`	Trajectory landmark shapes
`tlayout`	Trajectory layout (if existent)

Author(s)

Daniel C. Ellwanger

`addTrail` *ADD trail*

Description

Function to define a single trail on the trajectory.

Usage

```
addTrail(sce, from, to, name)
```

Arguments

`sce`	An object of class `SingleCellExperiment`
`from`	Start landmark
`to`	End landmark
`name`	Name of trail

Details

A trajectory can be composed of multiple single trails (e.g., developmental progression from a common start towards distinct terminal phenotypes). Start and endpoints of trails can be identified visually using the plot function `plotMap`. Here, start (=from) and end (=to) IDs of landmarks are starting with the character "B" (for branching points), "H" (for trail heads, i.e. terminal nodes), and "U" for user-defined landmarks.

Diagnostic messages

An error is thrown if the trajectory has not been fitted yet. Please, call `fitTrajectory` first. Further, an error is thrown if the provided start or end ID is unknown. A warning is shown if a trail with the same name already exists and gets re-defined.

Value

An updated object of class `SingleCellExperiment`

Author(s)

Daniel C. Ellwanger

See Also

`fitTrajectory` `landmarks` `plotMap`

Examples

```
# Example data
data(exSCE)

# Add trail
exSCE <- addTrail(exSCE, "H1", "H2", "Tr3")
trailNames(exSCE)
phenoNames(exSCE)
```

`connectStates` *Connect trajectory states*

Description

Connects states using maximum interface scoring. For each state an interface score is defined by the relative distribution of states in its local l-neighborhood. A filter is applied to remove outliers (ie. false positive neighbors). States are spanned by maximizing the total interface score.

Usage

```
connectStates(sce, l = 10)
```

Arguments

`sce`	A `SingleCellExperiment` object
`l`	Neighborhood size (default: 10)

Details

CellTrails assumes that the arrangement of samples in the computed lower-dimensional latent space constitutes a trajectory. Therefore, CellTrails aims to place single samples along a maximum parsimony tree, which resembles a branching developmental continuum. Distances between samples in the latent space are computed using the Euclidean distance.

To avoid overfitting and to facilitate the accurate identification of bifurcations, CellTrails simplifies the problem. Analogous to the idea of a ‘broken-stick regression’, CellTrails groups the data and perform linear fits to separate trajectory segments, which are determined by the branching chronology of states. This leaves the optimization problem of finding the minimum number of associations between states while maximizing the total parsimony, which in theory can be solved by any minimum spanning tree algorithm. CellTrails adapts this concept by assuming that adjacent states should be located nearby and therefore share a relative high number of neighboring cells.

Each state defines a submatrix of samples that is composed of a distinct set of data vectors, i.e., each state is a distinct set of samples represented in the lower-dimensional space. For each state CellTrails identifies the *l*-nearest neighbors to each state's data vector and takes note of their state memberships and distances. This results in two vectors of length *l* times the state size (i.e., a vector with memberships and a vector with distances).

CellTrails removes spurious neighbors (outliers), whose distance to a state is greater than or equal to

$$e^{median(log(D))+MAD(log(D))}$$

where D is a matrix containing all collected l-nearest neighbor sample distances to any state in the latent space.

For each state CellTrails calculates the relative frequency on how often a state occurs in the neighborhood of a given state, which is refered to as the interface cardinality scores.

CellTrails implements a greedy algorithm to find the tree maximizing the total interface cardinality score, similar to a minimum spanning tree algorithm (Kruskal, 1956). In a nutshell, all interface cardinality scores are organized in a sorted linked list, and a graph with no edges, but k nodes (one for each state) is initialized. During each iteration the highest score is selected, removed from the list and its corresponding edge (connecting two states), if it is not introducing a cycle or is already existent, is added to the graph. The algorithm terminates if the size of the graph is *k*-1 (with *k* equals number of states) or the list is empty. A cycle is determined if nodes were revisited while traversing the graph using depth-first search. Its construction has a relaxed requirement (number of edges < number of nodes) compared to a tree (number of edges = number of nodes - 1), which may result in a graph (forest) having multiple tree components, i.e. several trajectories or isolated nodes.

Diagnostic messages

An error is thrown if the states have not been defined yet; function `findStates` needs to be called first.

Value

An updated `SingleCellExperiment` object

Author(s)

Daniel C. Ellwanger

References

Kruskal, J.B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. Proc Amer Math Soc 7, 48-50.

See Also

`findStates` `states`

Examples

```
# Example data
data(exSCE)

# Connect states
exSCE <- connectStates(exSCE, l=30)
```

`contrastTrailExpr` *Differential trail expression analysis*

Description

Comparison of feature expression dynamic between two trails.

Usage

```
contrastTrailExpr(
  sce,
  feature_names = featureNames(sce),
  trail_names,
  score = "rmsd"
)
```

Arguments

`sce`	A `SingleCellExperiment` object
`feature_names`	Name of feature; can be multiple names
`trail_names`	Name of trails
`score`	Score type; one of {"rmsd", "tad", "abc", "cor"}

Details

Genes have non-uniform expression rates and each trail has a distinct set of upregulated genes, but also contains unequal numbers of cells. Because pseudotime is based on transcriptional change, its axis may be distorted, leading to stretched or compressed sections of longitudinal expression data that make comparison of trails challenging. To align different trails, despite these differences, CellTrails employs a dynamic programming based algorithm that has long been known in speech recognition, called dynamic time warping (Sakoe and Chiba, 1978). RNA expression rates are modeled analogous to speaking rates (Aach and Church, 2001); the latter accounts for innate non-linear variation in the length of individual phonemes (i.e., states) resulting in stretching and shrinking of word (i.e., trail) segments. This allows the computation of inter-trail alignment warps of individual expression time series that are similar but locally out of phase.

Univariate pairwise alignments are computed resulting in one warp per feature and per trail set. Similar to a (global) pairwise protein sequence alignment, monotonicity (i.e., no time loops) and continuity (i.e., no time leaps) constraints have to be imposed on the warping function to preserve temporal sequence ordering. To find the optimal warp, a recursion rule is applied which selects the local minimum of three moves through a dynamic programming matrix: suppose that query snapshot *g* and reference snapshot *h* have already been aligned, then the alignment of *h*+1 with *g*+1 is a (unit slope) diagonal move, *h* with *g*+1 denotes an expansion by repetition of *h*, and *h*+2 with *g*+1 contracts the query by dropping *h*+1.

The overall dissimilarity between two aligned expression time series *x* and *y* of length *n* is estimated by either the root-mean-square deviation $RMSD(x, y) = \sqrt(\sum(x - y)^2/n)$, the total aboslute deviation $TAD(x, y) = \sum(|x - y|)$, the area between the aligned dynamic curves (`ABC`), or Pearson's correlation coefficient (`cor`) over all aligned elements.

Value

Numeric value

Author(s)

Daniel C. Ellwanger

References

Sakoe, H., and Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. IEEE Transactions on Acoustics, Speech, and Signaling Processing 26, 43-49.

Aach, J., and Church, G.M. (2001). Aligning gene expression time series with time warping algorithms. Bioinformatics 17, 495-508.

See Also

`dtw`

Examples

```
# Example data
data(exSCE)

# Differential expression between trails
contrastTrailExpr(exSCE, feature_name=c("feature_1", "feature_10"),
                  trail_names=c("Tr1", "Tr2"), score="rmsd")
```

`embedSamples` *Spectral embedding of biological samples*

Description

Non-linear learning of a data representation that captures the intrinsic geometry of the trajectory. This function performs spectral decomposition of a graph encoding conditional entropy-based sample-to-sample similarities.

Usage

```
embedSamples(x, design = NULL)

## S4 method for signature 'matrix'
embedSamples(x, design = NULL)
```

Arguments

`x`	A `SingleCellExperiment` object or a numeric matrix with samples in columns and features in rows
`design`	A numeric matrix describing the factors that should be blocked

Details

Single-cell gene expression measurements comprise high-dimensional data of large volume, i.e. many features (e.g., genes) are measured in many samples (e.g., cells); or more formally, m samples can be described by the expression of n features (i.e., n dimensions). The cells' expression profiles are shaped by many distinct unobserved biological causes related to each cell's geno- and phenotype, such as developmental age, tissue region of origin, cell cycle stage, as well as extrinsic sources such as status of signaling receptors, and environmental stressors, but also technical noise. In other words, a single dimension, despite just containing gene expression information, represents an underlying combination of multiple dependent and independent, relevant and non-relevant factors, whereat each factors' individual contribution is non-uniform. To obtain a better resolution and to extract underlying information, CellTrails aims to find a meaningful low-dimensional structure - a manifold - that represents cells mainly by their temporal relation along a biological process.

This method assumes that the expression vectors are lying on or near a manifold with dimensionality d that is embedded in the n-dimensional space. By using spectral embedding CellTrails aims to amplify latent temporal information; it reduces noise (ie. truncates non-relevant dimensions) by transforming the expression matrix into a new dataset while retaining the geometry of the original dataset as much as possible.CellTrails captures overall cell-to-cell relations based on the statistical mutual dependency between any two data vectors. A high dependency between two samples should be represented by their close proximity in the lower-dimensional space.

First, the mutual depencency between samples is scored using mutual information. This entropy framework naturally requires discretization of data vectors by an indicator function, which assigns each continuous data point (expression value) to exactly one discrete interval (e.g. low, mid or high). However, measurement points located close to the interval borders may get wrongly assigned due to noise-induced fluctuations. Therefore, CellTrails fuzzifies the indicator function by using a piecewise polynomial function, i.e. the domain of each sample expression vector is divided into contiguous intervals (based on Daub *et al.*, 2004). Second, the computed mutual information matrix, which is left-bounded and composed of bits, is scaled to a generalized correlation coefficient. Third, CellTrails constructs a simple complete graph with *m* nodes, one for each data vector (ie. sample), and weights each edge between two nodes by a heat kernel function applied on the generalzied correlation coefficient. Finally, nonlinear spectral embedding (ie. spectral decomposition of the graph's adjacency matrix) is performed (Belkin & Niyogi, 2003; Sussman *et al.*, 2012) unfolding the manifold. Please note that this methods only uses the set of defined trajectory features in a `SingleCellExperiment` object; spike-in controls are ignored and are not listed as trajectory features.

To account for systematic bias in the expression data (e.g., cell cycle effects), a design matrix can be provided for the learning process. It should list the factors that should be blocked and their values per sample. It is suggested to construct a design matrix with `model.matrix`.

Diagnostic messages

The method throws an error if expression matrix contains samples with zero entropy (e.g., the samples exclusively contain non-detects, that is all expression values are zero).

Value

A list containing the following components:

`eigenvectors`	Ordered components of latent space
`eigenvalues`	Information content of latent components

Author(s)

Daniel C. Ellwanger

References

Daub, C.O., Steuer, R., Selbig, J., and Kloska, S. (2004). Estimating mutual information using B-spline functions – an improved similarity measure for analysing gene expression data. BMC Bioinformatics 5, 118.

Belkin, M., and Niyogi, P. (2003). Laplacian eigenmaps for dimensionality reduction and data representation. Neural computation 15, 1373-1396.

Sussman, D.L., Tang, M., Fishkind, D.E., and Priebe, C.E. (2012). A Consistent Adjacency Spectral Embedding for Stochastic Blockmodel Graphs. J Am Stat Assoc 107, 1119-1128.

See Also

`SingleCellExperiment` `trajectoryFeatureNames` `model.matrix`

Examples

```
# Example data
data(exSCE)

# Embed samples
res <- embedSamples(exSCE)
```

`enrichment.test` *Enrichment test*

Description

Statistical enrichment analysis using either a Hypergeometric or Fisher's test

Usage

```
enrichment.test(
  sample_true,
  sample_size,
  pop_true,
  pop_size,
  method = c("fisher", "hyper")
)
```

Arguments

`sample_true`	Number of hits in sample
`sample_size`	Size of sample
`pop_true`	Number of hits in population
`pop_size`	Size of population
`method`	Statistical method that should be used

Details

Hypergeometric or one-tailed Fisher exact test is useful for enrichment analyses. For example, one needs to estimate which features are enriched among a set of instances sampled from a population.

Value

A list containing the following components:

`p.value`	P-value of the test
`odds.ratio`	Odds ratio
`conf.int`	Confidence interval for the odds ratio (only shown with method="fisher")
`method`	Used statistical test

Author(s)

Daniel C. Ellwanger

See Also

`Hypergeometric` and `fisher.test`

Examples

```
# Population has 13 of total 52 instances positive for a given feature
# Sample has 1 of total 5 instances positive for a given feature
# Test for significance of enrichment in sample
enrichment.test(sample_true=1, sample_size=5,
                pop_true=13, pop_size=52, method="fisher")
```

---

exSCE *Example single-cell expression data*

Description

This dataset contains simulated transcript expression profiles of 25 genes in expressed 100 cells. Simulation was performed using using the Negative Binomial Distribution. Distribution parameters for each feature were sampled from a Gamma distribution. The resulting expression matrix is log2-scaled and was stored in in an object of class 'SingleCellExperiment' (assay `logcounts`). The sample metainformation contains the underlying (discrete) simulated age of the cells.

Usage

```
data(exSCE)
```

Format

An object of class `SingleCellExperiment`

---

featureNames,SingleCellExperiment-method *GET feature names*

Description

Retrieve feature names from a `SingleCellExperiment` object

Usage

```
## S4 method for signature 'SingleCellExperiment'
featureNames(object)
```

Arguments

`object`	An object of class `SingleCellExperiment`

Details

Wrapper for `rownames(object)`

Value

A `character` vector

Author(s)

Daniel C. Ellwanger

See Also

`SingleCellExperiment`

Examples

```
# Example data
data(exSCE)

featureNames(exSCE)
```

---

filterTrajFeaturesByCOV   *Filter features by Coefficient of Variation (COV)*

Description

Filters trajectory features by their coefficient of variation.

Usage

```
filterTrajFeaturesByCOV(sce, threshold, design = NULL, show_plot = TRUE)
```

Arguments

`sce`	An `SingleCellExperiment` object
`threshold`	Minimum coefficient of variation; numeric value between 0 and 1
`design`	A numeric matrix describing the factors that should be blocked
`show_plot`	Indicates if plot should be shown (default: TRUE)

Details

For each trajectory feature x listed in the `SingleCellExperiment` object the coefficient of variation is computed by $CoV(x) = sd(x)/mean(x)$. Features with a CoV(x) greater than `threshold` remain labeled as trajectory feature in the `SingleCellExperiment` object, otherwise they are not considered for dimensionality reduction, clustering and trajectory reconstruction. Please note that spike-in controls are ignored and are not listed as trajectory features.

To account for systematic bias in the expression data (e.g., cell cycle effects), a design matrix can be provided for the learning process. It should list the factors that should be blocked and their values per sample. It is suggested to construct a design matrix with `model.matrix`.

Value

A `character` vector

Author(s)

Daniel C. Ellwanger

See Also

`trajFeatureNames` `model.matrix`

Examples

```
# Simulate example data
set.seed(1101)
dat <- simulate_exprs(n_features=15000, n_samples=100)

# Create container
alist <- list(logcounts=dat)
sce <- SingleCellExperiment(assays=alist)

# Filter incrementally
trajFeatureNames(sce) <- filterTrajFeaturesByDL(sce, threshold=2)
trajFeatureNames(sce) <- filterTrajFeaturesByCOV(sce, threshold=0.5)

# Number of features
length(trajFeatureNames(sce)) #filtered
nrow(sce) #total
```

---

filterTrajFeaturesByDL

Filter trajectory features by Detection Level (DL)

Description

Filters trajectory features that are detected in a minimum number of samples.

Usage

```
filterTrajFeaturesByDL(sce, threshold, show_plot = TRUE)
```

Arguments

`sce`	An `SingleCellExperiment` object
`threshold`	Minimum number of samples; if value < 1 it is interpreted as fraction, otherwise as absolute sample count
`show_plot`	Indicates if plot should be shown (default: TRUE)

Details

The detection level denotes the fraction of samples in which a feature was detected. For each trajectory feature listed in the CellTrailsSet object the relative number of samples having a feature expression value greater than 0 is counted. Features that are expressed in a fraction of all samples greater than `threshold` remain labeled as trajectory feature as listed in the `SingleCellExperiment` object, otherwise they may be not considered for dimensionality reduction, clustering, and trajectory reconstruction. If the parameter `threshold` fullfills `threshold` $>= 1$ it becomes converted to a relative fraction of the total sample count. Please note that spike-in controls are ignored and are not listed as trajectory features.

Value

A `character` vector

Author(s)

Daniel C. Ellwanger

See Also

`trajFeatureNames`

Examples

```
# Example data
set.seed(1101)
dat <- simulate_exprs(n_features=15000, n_samples=100)

# Create container
alist <- list(logcounts=dat)
sce <- SingleCellExperiment(assays=alist)

# Filter features
tfeat <- filterTrajFeaturesByDL(sce, threshold=2)
head(tfeat)

# Set trajectory features to object
trajFeatureNames(sce) <- tfeat
```

```
# Number of features
length(trajFeatureNames(sce)) #filtered
nrow(sce) #total
```

---

filterTrajFeaturesByFF

Filter features by Fano Factor

Description

Filters trajectory features that exhibit a significantly high fano factor (index of dispersion) by considering average expression levels.

Usage

```
filterTrajFeaturesByFF(
  sce,
  threshold = 1.7,
  min_expr = 0,
  design = NULL,
  show_plot = TRUE
)
```

Arguments

`sce`	An `SingleCellExperiment` object
`threshold`	A Z-score cutoff (default: 1.7)
`min_expr`	Minimum average expression of feature to be considered
`design`	A numeric matrix describing the factors that should be blocked
`show_plot`	Indicates if plot should be shown (default: TRUE)

Details

To identify the most variable features an unsupervised strategy that controls for the relationship between a features's average expression intensity and its expression variability is applied. Features are placed into 20 bins based on their mean expression. For each bin the fano factor (a windowed version of the index of dispersion, IOD = variance / mean) distribution is computed and standardized (Z-score(x) = x/sd(x) - mean(x)/sd(x)). Features with a Z-score greater than `threshold` remain labeled as trajectory feature in the `SingleCellExperiment` object. The parameter `min_expr` defines the minimum average expression level of a feature to be considered for this filter method. Please note that spike-in controls are ignored and are not listed as trajectory features.

To account for systematic bias in the expression data (e.g., cell cycle effects), a design matrix can be provided for the learning process. It should list the factors that should be blocked and their values per sample. It is suggested to construct a design matrix with `model.matrix`.

Value

A `character` vector

Author(s)

Daniel C. Ellwanger

See Also

`trajFeatureNames` `model.matrix`

Examples

```
# Simulate example data
set.seed(1101)
dat <- simulate_exprs(n_features=15000, n_samples=100)

# Create container
alist <- list(logcounts=dat)
sce <- SingleCellExperiment(assays=alist)

# Filter incrementally
trajFeatureNames(sce) <- filterTrajFeaturesByDL(sce, threshold=2)
trajFeatureNames(sce) <- filterTrajFeaturesByCOV(sce, threshold=0.5)
trajFeatureNames(sce) <- filterTrajFeaturesByFF(sce, threshold=1.7)

# Number of features
length(trajFeatureNames(sce)) #filtered
nrow(sce) #total
```

`findSpectrum` *Determine number of informative latent dimensions*

Description

Identifies the dimensionality of the latent space

Usage

```
findSpectrum(x, frac = 100)
```

Arguments

`x`	A numeric vector with eigenvalues
`frac`	Fraction or number (if `frac` > 1) of eigengaps used to perform linear fit. (default: 100)

Details

Similar to a scree plot, this method generates a simple line segement plot showing the lagged differences between ordered eigenvalues (eigengaps). A linear fit is calucated on a fraction of top ranked values to identify informative eigenvectors.

Value

A `numeric` vector with indices of relevant dimensions

Author(s)

Daniel C. Ellwanger

See Also

`pca` `embedSamples`

Examples

```
# Example data
data(exSCE)

# Embedding
res <- embedSamples(exSCE)

# Find spectrum
d <- findSpectrum(res$eigenvalues, frac=30)
d
```

`findStates` *Identify trajectory states*

Description

Determines states using hierarchical spectral clustering with a *post-hoc* test.

Usage

```
findStates(sce, min_size = 0.01, min_feat = 5, max_pval = 1e-04, min_fc = 2)
```

Arguments

`sce`	A `SingleCellExperiment` object
`min_size`	The initial cluster dedrogram is cut at an height such that the minimum cluster size is at least `min_size`; if `min_size` < 1 than the fraction of total samples is used, otherwise it is used as absoulte count (default: 0.01).

`min_feat`	Minimum number of differentially expressed features between siblings. If this number is not reached, two neighboring clusters (siblings) in the pruned dendrogram get joined. (default: 5)
`max_pval`	Maximum *P*-value for differential expression computation. (default: 1e-4)
`min_fc`	Mimimum fold-change for differential expression computation (default: 2)

Details

To identify cellular subpopulations, CellTrails performs hierarchical clustering via minimization of a square error criterion (Ward, 1963) in the lower-dimensional space. To determine the cardinality of the clustering, CellTrails conducts an unsupervised *post-hoc* analysis. Here, it is assumed that differential expression of assayed features determines distinct cellular stages. First, Celltrails identifies the maximal fragmentation of the data space, i.e. the lowest cutting height in the clustering dendrogram that ensured that the resulting clusters contained at least a certain fraction of samples. Then, processing from this height towards the root, CellTrails iteratively joins siblings if they did not have at least a certain number of differentially expressed features. Statistical significance is tested by means of a two-sample non-parametric linear rank test accounting for censored values (Peto & Peto, 1972). The null hypothesis is rejected using the Benjamini-Hochberg (Benjamini & Hochberg, 1995) procedure for a given significance level.
Since this methods performs pairwise comparisons, the fold change threshold value is valid in both directions: higher and lower expressed than `min_fc`. Thus, input values < 0 are interpreted as a fold-change of 0. For example, `min_fc=2` checks for features that are 2-fold differentially expressed in two given states (e.g., S1, S2). Thus, a feature can be either 2-fold higher expressed in state S1 or two-fold lower expressed in state S2 to be validated as differentially expressed.
Please note that this methods only uses the set of defined trajectory features in a `SingleCellExperiment` object; spike-in controls are ignored and are not listed as trajectory features.

Diagnostic messages

An error is thrown if the samples stored in the `SingleCellExperiment` object were not embedded yet (ie. the `SingleCellExperiment` object does not contain a latent space matrix object; `latentSpace(object)`is `NULL`).

Value

A `factor` vector

Author(s)

Daniel C. Ellwanger

References

Ward, J.H. (1963). Hierarchical Grouping to Optimize an Objective Function. Journal of the American Statistical Association, 58, 236-244.

Peto, R., and Peto, J. (1972). Asymptotically Efficient Rank Invariant Test Procedures (with Discussion). Journal of the Royal Statistical Society of London, Series A 135, 185–206.

Benjamini, Y., and Hochberg, Y. (1995). Controlling the false discovery rate: a practical and powerful approach to multiple testing. Journal of the Royal Statistical Society Series B 57, 289–300.

See Also

`latentSpace` `trajectoryFeatureNames`

Examples

```
# Example data
data(exSCE)

# Find states
cl <- findStates(exSCE, min_feat=2)
head(cl)
```

`fitDynamic` *Fit expression dynamic*

Description

Fits feature expression as a function of pseudotime along a defined trail.

Usage

```
fitDynamic(sce, feature_name, trail_name)
```

Arguments

`sce`	A `SingleCellExperiment` object
`feature_name`	Name of feature
`trail_name`	Name of trail

Details

A trail is an induced subgraph of the trajectory graph. A trajectory graph is composed of samples (nodes) that are connected (by weighted edges) if they are chronologically related. A trail has to be defined by the user using `addTrail`. A pseudotime vector is extracted by computing the geodesic distance for each sample from the trail's start node. To infer the expression level of a feature as a function of pseudotime, CellTrails used generalized additive models with a single smoothing term with four basis dimensions. Here, for each feature CellTrails introduces prior weights for each observation to lower the confounding effect of drop-outs to the maximum-likelihood-based fitting process as follows. Each non-detect of feature j in state h is weighted by the relative fraction of non-detects of feature j in state h; detected values are always assigned weight = 1.

Value

An object of type `list` with the following components

`pseudotime`	The pseudotime along the trail
`expression`	The fitted expression values for each value of pseudotime
`gam`	A object of class `gamObject`

Author(s)

Daniel C. Ellwanger

See Also

`addTrail` `gamObject`

Examples

```
# Example data
data(exSCE)

# Fit dynamic
fit <- fitDynamic(exSCE, feature_name="feature_3", trail_name="Tr1")

summary(fit)
```

`fitTrajectory` *Align samples to trajectory*

Description

Orthogonal projection of each sample to the trajectory backbone.

Usage

```
fitTrajectory(sce)
```

Arguments

`sce`	A `SingleCellExperiment` object

Details

The previously selected component (with *k* states) defines the trajectory backbone. With this function CellTrails embeds the trajectory structure in the latent space by computing *k*-1 straight lines passing through *k* mediancentres (Bedall & Zimmermann, 1979) of adjacent states. Then, a fitting function is learned. Each sample is projected to its most proximal straight line passing through the mediancentre of its assigned state. Here, whenever possible, projections on line segments *between* two mediancentres are preferred. Residuals (fitting deviations) are given by the Euclidean distance between the sample's location and the straight line. Finally, a weighted acyclic trajectory graph can be constructed based on each sample's position along its straight line. In addition, data vectors are connected to mediancentres to enable the proper determination of branching points. Each edge is weighted by the distance between each node (sample) after orthogonal projection.

Of note, the fitting function implies potential side branches in the trajectory graph; those could be caused due to technical variance or encompass samples that were statistically indistinguishable from the main trajectory given the selected genes used for trajectory reconstruction.

Diagnostic messages

An error is thrown if an trajectory graph component was not computed or selected yet; functions `connectStates` and `selectTrajectory` need to be run first.

Value

An updated `SingleCellExperiment` object

Author(s)

Daniel C. Ellwanger

References

Bedall, F.K., and Zimmermann, H. (1979). Algorithm AS143. The mediancentre. Appl Statist 28, 325-328.

See Also

`connectStates` `selectTrajectory`

Examples

```
# Example data
data(exSCE)

# Align samples to trajectory
exSCE <- fitTrajectory(exSCE)
```

`landmarks` *GET landmarks*

Description

Gets landmarks from a `SingleCellExperiment` object.

Usage

```
landmarks(object)
```

Arguments

`object`	A `SingleCellExperiment` object

Details

Trail branches (B) and heads (H) are automatically assigned; landmarks can also be defined on the trajectory by the user (U). Landmarks can be used to extract single trails from a trajectory.

Value

A character vector with sample names

Author(s)

Daniel C. Ellwanger

See Also

`userLandmarks`

Examples

```
# Example data
data(exSCE)

# Get landmarks
landmarks(exSCE)[seq_len(5)]
```

`latentSpace` *GET CellTrails' latent space*

Description

Retrieve computed latent space from a `SingleCellExperiment` object.

Usage

```
latentSpace(object)
```

Arguments

`object`	A `SingleCellExperiment` object

Details

Returns the latent space set for a CellTrails analysis. The resulting matrix is numeric. Rows are samples and columns are *d* components. It is a wrapper for `reducedDim` to ensure that the proper matrix is received from a `SingleCellExperiment` object.

Value

An object of class `matrix`

Author(s)

Daniel C. Ellwanger

See Also

`SingleCellExperiment` `reducedDim`

Examples

```
# Example data
data(exSCE)

# Get latent space
latentSpace(exSCE)[seq_len(5), ]
```

`latentSpace<-` *SET latent space*

Description

Set CellTrails' latent space to a `SingleCellExperiment` object.

Usage

```
latentSpace(object) <- value
```

Arguments

`object`	A `SingleCellExperiment` object
`value`	A numeric matrix with rows are samples and columns are components

Details

Rows need to be samples and columns to be *d* components (spanning the lower-dimensional latent space).

Value

An updated object of class `SingleCellExperiment`

Author(s)

Daniel C. Ellwanger

See Also

`SingleCellExperiment` `reducedDim`

Examples

```
# Example data
data(exSCE)

# Set latent space
latentSpace(exSCE) <- pca(exSCE)$components[, seq_len(10)]
```

`manifold2D` *GET 2D manifold representation*

Description

Returns 2D manifold representation of latent space from `SingleCellExperiment` object

Usage

```
manifold2D(object)
```

Arguments

`object`	A `SingleCellExperiment` object

Value

A `numeric` matrix

Author(s)

Daniel C. Ellwanger

Examples

```
# Example data
data(exSCE)

manifold2D(exSCE)[seq_len(5), ]
```

`manifold2D<-` *SET 2D manifold representation*

Description

Stores 2D manifold representation in `SingleCellExperiment` object

Usage

```
manifold2D(object) <- value
```

Arguments

`object`	A `SingleCellExperiment` object
`value`	A `numeric` matrix with one column per dimension

Value

An updated object of class `SingleCellExperiment`

Author(s)

Daniel C. Ellwanger

Examples

```
# Example data
data(exSCE)

gp <- plotManifold(exSCE, color_by="featureName", name="feature_10",
                   recalculate=TRUE)
manifold2D(exSCE) <- gp
```

`pca` *Principal Component Analysis*

Description

Performs principal component analysis by spectral decomposition of a covariance or correlation matrix

Usage

```
pca(sce, do_scaling = TRUE, design = NULL)
```

Arguments

`sce`	`SingleCellExperiment` object
`do_scaling`	FALSE = covariance matrix, TRUE = correlation matrix
`design`	A numeric matrix describing the factors that should be blocked

Details

The calculation is done by a spectral decomposition of the (scaled) covariance matrix of the trajectory features as defined in the `SingleCellExperiment` object. Features with zero variance get automatically removed. Please note that this methods only uses the set of defined trajectory features in a `SingleCellExperiment` object; spike-in controls are ignored and are not listed as trajectory features.

To account for systematic bias in the expression data (e.g., cell cycle effects), a design matrix can be provided for the learning process. It should list the factors that should be blocked and their values per sample. It is suggested to construct a design matrix with `model.matrix`.

Value

A `list` object containing the following components:

`components`	Principal components
`eigenvalues`	Variance per component
`variance`	Fraction of variance explained by each component
`loadings`	Loading score for each feature

Author(s)

Daniel C. Ellwanger

See Also

`SingleCellExperiment` `model.matrix`

Examples

```
# Example data
data(exSCE)

# Principal component analysis
res <- pca(exSCE)

# Find relevant number of principal components
d <- findSpectrum(res$eigenvalues, frac=20)

barplot(res$variance[d] * 100, ylab="Variance (%)",
        names.arg=colnames(res$components)[d], las=2)
plot(res$component, xlab="PC1", ylab="PC2")
```

phenoNames *GET phenotype names*

Description

Retrieve phenotype names from a `SingleCellExperiment` object

Usage

```
phenoNames(object)
```

Arguments

`object`	An object of class `SingleCellExperiment`

Details

Wrapper for `colnames(colData(object))`

Value

A `character` vector

Author(s)

Daniel C. Ellwanger

See Also

`SingleCellExperiment`

Examples

```
# Example data
data(exSCE)

phenoNames(exSCE)
```

plotDynamic *Visualize dynamics*

Description

Shows dynamics of one or multiple features along a given trail

Usage

```
plotDynamic(sce, feature_name, trail_name)
```

Arguments

`sce`	A `SingleCellExperiment` object
`feature_name`	Name of one or multiple features
`trail_name`	Name of trail

Details

An error is thrown if the `trail_name` or `feature_name` are unknown. The function is case-sensitiv. All available trails can be listed by `trailNames`, all features with `featureNames`.

Value

A `ggplot` object

Author(s)

Daniel C. Ellwanger

See Also

`addTrail` `trailNames` `featureNames`

Examples

```
# Example data
data(exSCE)

# Plot dynamic of feature_10
plotDynamic(exSCE, trail_name="Tr1", feature_name="feature_1")
# Plot dynamic of feature_1 and feature_10
plotDynamic(exSCE, trail_name="Tr1",
            feature_name=c("feature_1", "feature_10"))
```

plotManifold *Visualize the learned manifold*

Description

Method visualizes an approximation of the manifold in the latent space in two dimensions.

Usage

```
plotManifold(
  sce,
  color_by = c("phenoName", "featureName"),
  name,
  perplexity = 30,
  recalculate = FALSE
)
```

Arguments

`sce`	A `SingleCellExperiment` object
`color_by`	Indicates if nodes are colorized by a feature expression ('featureName') or phenotype label ('phenoName')
`name`	A character string specifying the featureName or phenoName
`perplexity`	Perplexity parameter for tSNE computation (default: 30)
`recalculate`	Indicates if tSNE should be recalcuated and results returned (default: FALSE)

Details

Visualizes the learned lower-dimensional manifold in two dimensions using an approximation obtained by Barnes-Hut implementation of t-Distributed Stochastic Neighbor Embedding (tSNE; van der Maaten and Hinton 2008). Each point in this plot represents a sample. Points can be colorized according to feature expression or experimental metadata. The points' coloration can be defined via the attributes `color_by` and `name`, respectively. A previously computed tSNE visualization will be reused if it was set accordingly (see `manifold2D<-`). The parameter `perplexity` is used for the tSNE calculation.

Value

A `ggplot` object

Author(s)

Daniel C. Ellwanger

References

van der Maaten, L.J.P. & Hinton, G.E., 2008. Visualizing High-Dimensional Data Using t-SNE. Journal of Machine Learning Research, 9, pp.2579-2605.

See Also

`Rtsne` `latentSpace` `manifold2D`

Examples

```
# Example data
data(exSCE)

plotManifold(exSCE, color_by="featureName", name="feature_10")
gp <- plotManifold(exSCE, color_by="phenoName", name="age",
                   recalculate=TRUE)
manifold2D(exSCE) <- gp
```

---

plotMap *Visualize expression maps*

Description

Method visualizes topographical expression maps in two dimensions.

Usage

```
plotMap(
  sce,
  color_by = c("phenoName", "featureName"),
  name,
  type = c("surface.fit", "surface.se", "raw"),
  samples_only = FALSE
)
```

Arguments

`sce`	A `SingleCellExperiment` object
`color_by`	Indicates if nodes are colorized by a feature expression
`name`	A character string specifying the featureName or phenoName
`type`	Type of map; one of "raw","surface.fit","surface.se"
`samples_only`	If only individual samples should be colorized rather than the whole surface (default: FALSE)

Details

Two-dimensional visualization of the trajectory. The red line representsthe trajectory and individual points denote samples. This plot type can either show thetopography of a given feature's expression landscape or colorizes individual samples by a metadata label. The feature is selected by setting the parameter `color_type` and the respecitve `name`. To show feature expression, a surface is fitted using isotropic (i.e., same parameters for both map dimensions) thin-plate spline smoothing in `gam`. It gives an overview of expression dynamics along all branches of the trajectory. The parameter `type` defines if either the raw/original expression data shoud be shown, the full fitted expression surface should be shown (`type="surface.fit"`) or the standard error of the surface prediction (`type="surface.se"`), or the expression values of single samples only (`type="surface.fit"` and `only_samples=TRUE`).

To show all landmarks on the map, please use the parameters `color_by="phenoName"` and `name="landmark"`.

Value

A `ggplot` object

Author(s)

Daniel C. Ellwanger

See Also

`gam`

Examples

```
# Example data
data(exSCE)

# Plot landmarks
plotMap(exSCE, color_by="phenoName", name="landmark")

# Plot phenotype
plotMap(exSCE, color_by="phenoName", name="age")

# Plot feature expression map
plotMap(exSCE, color_by="featureName", name="feature_10", type="surface.fit")
plotMap(exSCE, color_by="featureName", name="feature_10", type="surface.fit",
        samples_only=TRUE)

#Plot surface fit standard errors
plotMap(exSCE, color_by="featureName", name="feature_10", type="surface.se")
```

`plotStateExpression` *Visualize feature expression distribution per state*

Description

Violin plots showing the expression distribution of a feature per state.

Usage

```
plotStateExpression(sce, feature_name)
```

Arguments

`sce`	A `SingleCellExperiment` object
`feature_name`	The name of the feature to be visualized

Details

Each data point displays the feature's expression value in a single sample. A violine plot shows the density (mirrored on the y-axis) of the expression distribution per sample.

Value

A `ggplot` object

Author(s)

Daniel C. Ellwanger

See Also

`ggplot` `states`

Examples

```
# Example data
data(exSCE)

plotStateExpression(exSCE, feature_name="feature_1")
```

`plotStateSize` *Visualize the number of samples per state*

Description

Shows barplot of state size distribution

Usage

```
plotStateSize(sce)
```

Arguments

`sce`	A `SingleCellExperiment` object

Details

Barplot showing the absolute number of samples per state.

Value

A `ggplot` object

Author(s)

Daniel C. Ellwanger

See Also

`ggplot` `states`

Examples

```
# Example data
data(exSCE)

plotStateSize(exSCE)
```

`plotStateTrajectory` *Visualize state trajectory graph*

Description

Method visualizes the state-to-state relations delineating the trajectory backbone.

Usage

```
plotStateTrajectory(
  sce,
  color_by = c("phenoName", "featureName"),
  name,
  component = NULL,
  point_size = 3,
  label_offset = 2,
  recalculate = FALSE
)
```

Arguments

`sce`	A `SingleCellExperiment` object
`color_by`	Indicates if nodes are colorized by a feature expression ('featureName') or phenotype label ('phenoName')
`name`	A character string specifying the featureName or phenoName
`component`	Component of trajectory graph that should be shown (integer value)
`point_size`	Adjusts the point size of the data points shown
`label_offset`	Adjusts the offset of the data point labels
`recalculate`	If layout should be re-drawn (default: FALSE)

Details

Shows a single tree component of the computed trajectory graph. Each point in this plot represents a state and can be colorized according to feature expression (mean expression per state) or experimental metadata (arithmetic mean or percentage distribution of categorial values). The component is defined by parameter `component`. If the trajectory graph contains only a single component, then this parameter can be left undefined. The points' coloration can be defined via the attributes `color_by` and `name`, respectively. Missing sample lables are recovered using nearest neighbor learning.
If the state trajectory graph layout was set with `stateTrajLayout<-` then the layout will be reused for visualization.

Value

A `ggplot` object

Author(s)

Daniel C. Ellwanger

See Also

`connectStates`

Examples

```
# Example data
data(exSCE)

plotStateTrajectory(exSCE, color_by="phenoName", name="age",
                    component=1, point_size = 1.5, label_offset = 4)

gp <- plotStateTrajectory(exSCE, color_by="featureName", name="feature_1",
                          component=1, recalculate=TRUE)
stateTrajLayout(exSCE) <- gp
```

`plotTrail` *Visualize single trails*

Description

Method highlights a single trail on the trajectory map

Usage

```
plotTrail(sce, name)
```

Arguments

`sce`	A `SingleCellExperiment` object
`name`	Name of the trail

Details

A trail can be defined with the function `addTrail` between two landmarks. User-defined landmarks can be set with the function `userLandmarks`. This function visualizes the start and endpoints, and the pseudotime of a defined trail along the trajectory. The trail pseudotimes can be directly accessed via the `trails`.

An error is thrown if the `trail_name` is unknown. The function is case-sensitiv. All available trails can be listed by `trailNames`.

Value

A `ggplot` object

Author(s)

Daniel C. Ellwanger

See Also

`addTrail` `userLandmarks` `trailNames` `trails`

Examples

```
# Example data
data(exSCE)

# Plot trail
plotTrail(exSCE, name="Tr1")
```

`plotTrajectoryFit` *Visualize trajectory fit residuals*

Description

Method visualizes the fitting residuals along the trajectory backbone.

Usage

```
plotTrajectoryFit(sce)
```

Arguments

`sce`	A `SingleCellExperiment` object

Details

Shows the trajectory backbone (longest shortest path between two samples) and the fitting deviations of each sample indicated by the perpendicular jitter. Data points are colorized by state.

Value

A `ggplot` object

Author(s)

Daniel C. Ellwanger

See Also

`fitTrajectory` `trajResiduals`

Examples

```
# Example data
data(exSCE)

plotTrajectoryFit(exSCE)
```

`read.ygraphml` *Reads trajectory graph layout*

Description

Reads ygraphml file containing the trajectory graph's layout

Usage

```
read.ygraphml(file)
```

Arguments

`file`	A character string naming a file

Details

To visualize the trajectory graph, a proper graph layout has to be computed. Ideally, edges should not cross and nodes should not overlap. CellTrails enables the export and import of the trajectory graph structure using the graphml file format. This file format can be interpreted by most third-party graph analysis applications, allowing the user to subject the trajectory graph to a wide range of layout algorithms. Please note that the graphml file needs to contain layout information ("<y:Geometry x=... y=... >" entries) as provided by the 'ygraphml' file definition used by the Graph Visualization Software 'yEd' (freely available from yWorks GmbH, http://www.yworks.com/products/yed).

Value

An `data.frame` with coordinates of data points and visualization metadata

Author(s)

Daniel C. Ellwanger

See Also

`write.ygraphml`

Examples

```
# Example data
data(exSCE)

## Not run:
fn <- system.file("exdata", "exDat.graphml", package="CellTrails")
tl <- read.ygraphml(fn)

## End(Not run)
```

`removeTrail` *REMOVE trail*

Description

Removes trail from a `SingleCellExperiment` object.

Usage

```
removeTrail(sce, name)
```

Arguments

`sce`	An object of class `SingleCellExperiment`
`name`	Name of trail

Details

Diagnostic messages

An error is thrown if the trail name is unknown. All stored trail names can be shown using function `trailNames`.

Value

An updated object of class `SingleCellExperiment`

Author(s)

Daniel C. Ellwanger

See Also

`trailNames` `addTrail`

Examples

```
# Example data
data(exSCE)

# Remove trail
trailNames(exSCE)
exSCE <- removeTrail(exSCE, "Tr1")
trailNames(exSCE)
```

---

sampleNames,SingleCellExperiment-method
GET sample names

Description

Retrieve sample names from a `SingleCellExperiment` object

Usage

```
## S4 method for signature 'SingleCellExperiment'
sampleNames(object)
```

Arguments

`object`	An object of class `SingleCellExperiment`

Details

Wrapper for `colnames(object)`

Value

A `character` vector

Author(s)

Daniel C. Ellwanger

See Also

`SingleCellExperiment`

Examples

```
# Example data
data(exSCE)

sampleNames(exSCE)[seq_len(5)]
```

`selectTrajectory` *Select component from trajectory graph*

Description

Retains a single component of a trajectory graph.

Usage

```
selectTrajectory(sce, component)
```

Arguments

`sce`	A `SingleCellExperiment` object
`component`	Number of component to be selected

Details

The construction of a trajectory graph may result in a forest having multiple tree components, which may represent individual trajectories or isolated nodes. This method should be used to extract a single component from the graph. A component is identified by its (integer) number.

Diagnostic messages

An error is thrown if the states have not been connected yet; function `connectStates` needs to be called first. An error is thrown if an unknown component (number) is selected.

Value

An updated `SingleCellExperiment` object

Author(s)

Daniel C. Ellwanger

See Also

`connectStates`

`findStates` `states`

Examples

```
# Example data
data(exSCE)

# Select trajectory
exSCE <- selectTrajectory(exSCE, component=1)
```

`showTrajInfo` *Shows relevant content of a SingleCellExperiment object for a CellTrails analysis*

Description

Shows relevant content of a SingleCellExperiment object for a CellTrails analysis

Usage

```
showTrajInfo(object)
```

Arguments

`object`	A `SingleCellExperiment` object

Value

`showTrajInfo` returns an invisible `NULL`

Author(s)

Daniel C. Ellwanger

Examples

```
# Example data
data(exSCE)

showTrajInfo(exSCE)
```

`simulate_exprs` *Simulation of RNA-Seq expression data*

Description

Simple simulation of RNA-Seq expression data estimating counts based on the negative binomial distribution

Usage

```
simulate_exprs(n_features, n_samples, prefix_sample = "")
```

Arguments

`n_features`	Number of genes
`n_samples`	Number of samples
`prefix_sample`	Prefix of sample name

Details

RNA-Seq counts are generated using the Negative Binomial Distribution. Distribution parameters for each feature are sampled from a Gamma distribution. The resulting expression matrix is log2-scaled.

Value

A numeric matrix with genes in rows and samples in columns

Author(s)

Daniel C. Ellwanger

See Also

`NegBinomial` and `GammaDist`

Examples

```
# Matrix with 100 genes and 50 cells
dat <- simulate_exprs(n_features=100, n_samples=50)
```

`states` *GET states*

Description

Retrieve computed states from a `SingleCellExperiment` object

Usage

```
states(object)
```

Arguments

`object`	An object of class `SingleCellExperiment`

Details

State information is extracted from `colData`; factor levels are alphanumerically ordered by ID.

Value

A `factor` vector

Author(s)

Daniel C. Ellwanger

See Also

`SingleCellExperiment` `findStates`

Examples

```
# Example data
data(exSCE)

states(exSCE)[seq_len(5)]
```

`states<-` *SET states*

Description

Sets states to a `SingleCellExperiment` object

Usage

```
states(object) <- value
```

Arguments

`object`	An object of class `SingleCellExperiment`
`value`	A numeric, character or factor vector

Details

State information is added to a `SingleCellExperiment` object via `colData`. If the vector containing the cluster assignments is numeric, the prefix "S" is added and the vector is converted to type factor.

Value

An updated object of class `SingleCellExperiment`

Author(s)

Daniel C. Ellwanger

See Also

colData

Examples

```
# Example data
data(exSCE)

# Assign clusters
cl <- kmeans(logcounts(exSCE), centers=10)$cluster
states(exSCE) <- cl
```

`stateTrajLayout<-` *SET state trajectory layout*

Description

Stores layout of state trajectory in `SingleCellExperiment` object

Usage

```
stateTrajLayout(object) <- value
```

Arguments

`object`	A `SingleCellExperiment` object
`value`	A `ggplot` object

Value

An updated object of class `SingleCellExperiment`

Author(s)

Daniel C. Ellwanger

Examples

```
# Example data
data(exSCE)

gp <- plotStateTrajectory(exSCE, color_by="featureName",
                          name="feature_10", recalculate=TRUE)
stateTrajLayout(exSCE) <- gp
```

`trailNames` *GET trail names*

Description

Function to extract trail names from `SingleCellExperiment` object.

Usage

```
trailNames(object)
```

Arguments

`object`	An object of class `SingleCellExperiment`

Value

A `character` vector

Author(s)

Daniel C. Ellwanger

See Also

`addTrail`

Examples

```
# Example data
data(exSCE)

trailNames(exSCE)
```

`trailNames<-` *SET trail names*

Description

Enables to rename trails stored in a `SingleCellExperiment` object.

Usage

```
trailNames(object) <- value
```

Arguments

`object`	An object of class `SingleCellExperiment`
`value`	A character vector with the trail names

Details

Diagnostic messages

An error is thrown if the number of names does not correspond to the number of trails stored in the object. Further, trail names are required to be unique.

Value

An updated object of class `SingleCellExperiment`

Author(s)

Daniel C. Ellwanger

See Also

`addTrail`

Examples

```
# Example data
data(exSCE)

trailNames(exSCE)
trailNames(exSCE) <- c("ABC", "DEF")
trailNames(exSCE)
```

`trails` *GET trails*

Description

Function to extract trail pseudotimes from a `SingleCellExperiment` object.

Usage

```
trails(object)
```

Arguments

`object`	An object of class `SingleCellExperiment`

Value

A DataFrame with `numeric` columns

Author(s)

Daniel C. Ellwanger

See Also

`addTrail`

Examples

```
# Example data
data(exSCE)

trails(exSCE)
```

`trajComponents` *GET trajectory component states*

Description

Returns states of trajectory components `SingleCellExperiment` object

Usage

```
trajComponents(object)
```

Arguments

`object`	A `SingleCellExperiment` object

Value

A `character` vector

Author(s)

Daniel C. Ellwanger

Examples

```
# Example data
data(exSCE)

trajComponents(exSCE)
```

`trajFeatureNames` *GET trajectory feature names*

Description

Retrieve names of features that were selected for trajectory reconstruction from a `SingleCellExperiment` object.

Usage

```
trajFeatureNames(object)
```

Arguments

`object`	An object of class `SingleCellExperiment`

Details

Features can be selected prior to trajectory inference. This method retrieves the user-defined features from a `SingleCellExperiment` object. The return value is a character vector containing the feature names.

Value

An object of class `character`

Author(s)

Daniel C. Ellwanger

Examples

```
# Example data
data(exSCE)

# Get trajectory features
trajFeatureNames(exSCE)[seq_len(5)]
```

`trajFeatureNames<-` *SET trajectory features by name*

Description

Function to set trajectory features by name

Usage

```
trajFeatureNames(object) <- value
```

Arguments

`object`	An object of class `SingleCellExperiment`
`value`	A character vector

Value

An updated object of class `SingleCellExperiment`

Author(s)

Daniel C. Ellwanger

Examples

```
# Example data
data(exSCE)

# Set trajectory features
trajFeatureNames(exSCE) <- rownames(exSCE)[seq_len(5)]
```

`trajLayout` *GET trajectory layout*

Description

Returns trajectory layout from `SingleCellExperiment` object

Usage

```
trajLayout(object)
```

Arguments

`object`	A `SingleCellExperiment` object

Value

A `data.frame`

Author(s)

Daniel C. Ellwanger

Examples

```
# Example data
data(exSCE)

trajLayout(exSCE)[seq_len(5), ]
```

`trajLayout<-` *SET trajectory layout*

Description

Sets layout used for trajectory visualization to a `SingleCellExperiment` object.

Usage

```
trajLayout(object, adjust) <- value
```

Arguments

`object`	An object of class `SingleCellExperiment`
`adjust`	Indicates if layout has to be adjusted such that edge lengths correlate to pseudotime (default: TRUE)
`value`	A data.frame with x- and y-coordinates for each sample (rows = samples, columns = coordinates)

Details

CellTrails implements a module which can incorporate pseudotime information into the the graph layout (activated via parameter `adjust`). Here, edge lengths between two nodes (samples) will then correspond to the inferred pseudotime that separates two samples along the trajectory.

Diagnostic messages

An error is thrown if the number of rows of the layout does not correspond to the number of trajectory samples or if the number of columns is less than 2, or if the row names do not correspond to `sampleNames`.

Value

An updated object of class `SingleCellExperiment`

Author(s)

Daniel C. Ellwanger

See Also

`write.ygraphml` `trajSampleNames`

Examples

```
# Example data
data(exSCE)
tl <- trajLayout(exSCE)

trajLayout(exSCE) <- tl
```

`trajResiduals` *GET trajectory fitting residuals*

Description

Returns trajectory fitting residuals from `SingleCellExperiment` object

Usage

```
trajResiduals(object)
```

Arguments

`object`	A `SingleCellExperiment` object

Details

The trajectory fitting deviation is defined as the vector rejection from a sample in the latent space to the trajectory backbone. The trajectory backbone is defined by a tree spanning all relevant states. Samples get orthogonally projected onto straight lines connecting related states. This function quantifies the distance between the actual positon of a sample in the latent space and its projectd position on the trajectory backbone. In other words, the higher the distance, the higher its deviation (residual) from the trajectory fit. This function returns all residuals for each projected sample. Residuals of samples which were exluded for trajectory reconstruction are `NA`.

Value

A `numeric` vector

Author(s)

Daniel C. Ellwanger

See Also

`fitTrajectory` `trajSampleNames`

Examples

```
# Example data
data(exSCE)

trajResiduals(exSCE)[seq_len(5)]
```

`trajSampleNames` *GET trajectory sample names*

Description

Retrieve names of samples that were aligned onto the trajectory from a `SingleCellExperiment` object.

Usage

```
trajSampleNames(object)
```

Arguments

`object`	An object of class `SingleCellExperiment`

Details

A trajectory graph can be initially a forest. Trajectory fitting is performed on one component. This function returns the names of the samples which are member of the selected component.

Value

An object of class `character`

Author(s)

Daniel C. Ellwanger

Examples

```
# Example data
data(exSCE)

# Get trajectory samples
trajSampleNames(exSCE)[seq_len(5)]
```

`userLandmarks` *GET user-defined landmarks*

Description

Gets user-defined landmarks from a `SingleCellExperiment` object.

Usage

```
userLandmarks(object)
```

Arguments

`object`	A `SingleCellExperiment` object

Details

Landmarks can be defined on the trajectory by the user with `userLandmarks`. Landmarks can be used to extract single trails from a trajectory.

Value

A character vector with sample names

Author(s)

Daniel C. Ellwanger

See Also

`SingleCellExperiment`

Examples

```
# Example data
data(exSCE)

# Get landmarks
userLandmarks(exSCE)
```

`userLandmarks<-` *SET user-defined landmarks*

Description

Set user-defined landmarks to a `SingleCellExperiment` object.

Usage

```
userLandmarks(object) <- value
```

Arguments

`object`	A `SingleCellExperiment` object
`value`	A character vector with sample names

Details

Landmarks can be defined on the trajectory and can be necessary to extract individual trails from a trajectory.

Diagnostic messages

An error is thrown if the trajectory has not been reconstructed yet.

Value

An updated `SingleCellExperiment` object

Author(s)

Daniel C. Ellwanger

See Also

`SingleCellExperiment` `fitTrajectory`

Examples

```
# Example data
data(exSCE)

# Set landmarks
userLandmarks(exSCE) <- colnames(exSCE)[5:7]
```

`write.ygraphml` *Export trajectory graph*

Description

Writes graphml file containing the trajectory graph's structure.

Usage

```
write.ygraphml(
  sce,
  file,
  color_by = c("phenoName", "featureName"),
  name,
  node_label = "state"
)
```

Arguments

`sce`	A `SingleCellExperiment` object
`file`	Character string naming a file
`color_by`	Indicates if nodes are colorized by a feature expression ('featureName') or phenotype label ('phenoName')
`name`	A character string specifying the featureName or phenoName
`node_label`	Defines the node label name (optional). Can be either set to the samples' states ('state') or the samples' names ('name').

Details

To visualize the trajectory graph, a proper graph layout has to be computed. Ideally, edges should not cross and nodes should not overlap (i.e., a planar embedding of the graph). CellTrails enables the export and import of the trajectory graph structure using the graphml file format. This file format can be interpreted by most third-party graph analysis applications, allowing the user to subject the trajectory graph to a wide range of (tree) layout algorithms. In particular, its format has additional ygraph attributes best suited to be used with the Graph Visualization Software 'yEd' which is freely available from yWorks GmbH (http://www.yworks.com/products/yed) for all major platforms.

The colors of the nodes can be defined by the parameters `color_by` and `name`. Please note that the trajectory landmarks are indicated by setting `color_by='phenoName'` and name='landmark'. States can be indicated by `color_by='phenoName'` and name='state'.

If a layout is already present in the provided `CellTrailsSet` object, the samples' coordinates will be listed in the graphml file.

Diagnostic messages

An error is thrown if the trajectory has not been computed yet; function `fitTrajectory` needs to be called first. Feature names and phenotype names get checked and will throw an error if not contained in the dataset. Please note, the parameter `name` is case-sensitive.

Value

`write.ygraphml` returns an invisible `NULL`

Author(s)

Daniel C. Ellwanger

See Also

`fitTrajectory` `featureNames` `phenoNames`

Examples

```
# Example data
data(exSCE)

## Not run:
# Export trajectory graph structure to graphml
# Color nodes by gene expression (e.g, feature_10)
write.ygraphml(sce, file="yourFilePath", color_by="featureName",
              name="feature_10")

# Color nodes by metadata (e.g., state) and
# label nodes by the (simulated) age of each sample
write.ygraphml(sce, file="yourFilePath", color_by="phenoName",
              name="state", node_label="age")

# Color and label nodes by landmark type and id
write.ygraphml(sce, file="yourFilePath", color_by="phenoName",
              name="landmark", node_label="landmark")
## End(Not run)
```

Index