

# Package ‘CellTrails’

September 22, 2019

**Type** Package

**Title** Reconstruction, visualization and analysis of branching trajectories

**Version** 1.3.0

**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, mapee, methods, mgcv, reshape2, Rtsne, stats, splines, SummarizedExperiment, utils

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

**RoxygenNote** 6.0.1

**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** master

**git\_last\_commit** f3306bc

**git\_last\_commit\_date** 2019-05-02

**Date/Publication** 2019-09-21

**Author** Daniel Ellwanger [aut, cre, cph]

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

## R topics documented:

addTrail . . . . .	3
connectStates . . . . .	4
contrastTrailExpr . . . . .	6
embedSamples . . . . .	7
enrichment.test . . . . .	9
exSCE . . . . .	10
featureNames,SingleCellExperiment-method . . . . .	11
filterTrajFeaturesByCOV . . . . .	12
filterTrajFeaturesByDL . . . . .	13
filterTrajFeaturesByFF . . . . .	14
findSpectrum . . . . .	15
findStates . . . . .	16
fitDynamic . . . . .	18
fitTrajectory . . . . .	19
landmarks . . . . .	20
latentSpace . . . . .	21
latentSpace<- . . . . .	22
manifold2D . . . . .	23
manifold2D<- . . . . .	23
pca . . . . .	24
phenoNames . . . . .	25
plotDynamic . . . . .	26
plotManifold . . . . .	27
plotMap . . . . .	28
plotStateExpression . . . . .	29
plotStateSize . . . . .	30
plotStateTrajectory . . . . .	31
plotTrail . . . . .	32
plotTrajectoryFit . . . . .	33
read.ygraphml . . . . .	34
removeTrail . . . . .	35
sampleNames,SingleCellExperiment-method . . . . .	36
selectTrajectory . . . . .	37
showTrajInfo . . . . .	38
simulate_exprs . . . . .	38
states . . . . .	39
states<- . . . . .	40
stateTrajLayout<- . . . . .	41
trailNames . . . . .	41
trailNames<- . . . . .	42
trails . . . . .	43
trajComponents . . . . .	44

<i>addTrail</i>	3
trajFeatureNames	44
trajFeatureNames<-	45
trajLayout	46
trajLayout<-	46
trajResiduals	47
trajSampleNames	48
userLandmarks	49
userLandmarks<-	50
write.ygraphml	51
<b>Index</b>	<b>53</b>

---

addTrail	<i>ADD trail</i>
----------	------------------

---

## 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	<i>Connect trajectory states</i>
---------------	----------------------------------

---

**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

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

where D is a matrix containing all collected 1-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 referred 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 absolute 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 dependency 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 generalized 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	<i>Enrichment test</i>
-----------------	------------------------

---

**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:

<code>p.value</code>	P-value of the test
<code>odds.ratio</code>	Odds ratio
<code>conf.int</code>	Confidence interval for the odds ratio (only shown with <code>method="fisher"</code> )
<code>method</code>	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**

<code>sce</code>	An <code>SingleCellExperiment</code> object
<code>threshold</code>	Minimum coefficient of variation; numeric value between 0 and 1
<code>design</code>	A numeric matrix describing the factors that should be blocked
<code>show_plot</code>	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` `isSpike` `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  $\text{threshold} \geq 1$  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 isSpike

**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,  $IOD = \text{variance} / \text{mean}$ ) distribution is computed and standardized ( $Z\text{-score}(x) = x/\text{sd}(x) - \text{mean}(x)/\text{sd}(x)$ ). 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` `isSpike` `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

<code>x</code>	A numeric vector with eigenvalues
<code>frac</code>	Fraction or number (if <code>frac &gt; 1</code> ) of eigengaps used to perform linear fit. (default: 100)

**Details**

Similar to a scree plot, this method generates a simple line segment plot showing the lagged differences between ordered eigenvalues (eigengaps). A linear fit is calculated 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 dendrogram 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 absolute 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 <i>P</i> -value for differential expression computation. (default: 1e-4)
min_fc	Minimum 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	<i>Fit expression dynamic</i>
------------	-------------------------------

---

### 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 <code>gamObject</code>

### 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	<i>Align samples to trajectory</i>
---------------	------------------------------------

---

**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	<i>GET CellTrails' latent space</i>
-------------	-------------------------------------

---

**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**

<code>object</code>	A <code>SingleCellExperiment</code> object
<code>value</code>	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	<i>GET 2D manifold representation</i>
------------	---------------------------------------

---

**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<-	<i>SET 2D manifold representation</i>
--------------	---------------------------------------

---

**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-sensitive. 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_10")
# Plot dynamic of feature_1 and feature_10
plotDynamic(exSCE, trail_name="Tr1",
            feature_name=c("feature_1", "feature_10"))
```

---

plotManifold	<i>Visualize the learned manifold</i>
--------------	---------------------------------------

---

### 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 recalculated 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 represents the trajectory and individual points denote samples. This plot type can either show the topography 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 respective 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 should 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	<i>Visualize the number of samples per state</i>
---------------	--

---

**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 labels 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	<i>Visualize single trails</i>
-----------	--------------------------------

---

## 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-sensitive. 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	<i>Visualize trajectory fit residuals</i>
-------------------	---

---

**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	<i>Reads trajectory graph layout</i>
---------------	--------------------------------------

---

**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	<i>REMOVE trail</i>
-------------	---------------------

---

**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	<i>Select component from trajectory graph</i>
------------------	---

---

**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	<i>Shows relevant content of a SingleCellExperiment object for a Cell-Trails analysis</i>
--------------	---

---

**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	<i>Simulation of RNA-Seq expression data</i>
----------------	--

---

**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
c1 <- kmeans(logcounts(exSCE), centers=10)$cluster
states(exSCE) <- c1
```



---

```
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<-	<i>SET trail names</i>
--------------	------------------------

---

**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.

`trajFeatureNames<-`

45

### **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**

<code>object</code>	An object of class <code>SingleCellExperiment</code>
<code>value</code>	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	<i>GET trajectory layout</i>
------------	------------------------------

---

**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<-	<i>SET trajectory layout</i>
--------------	------------------------------

---

**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 pseudo-time (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)
t1 <- trajLayout(exSCE)

trajLayout(exSCE) <- t1
```

---

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 position of a sample in the latent space and its projected 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 excluded 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	<i>GET trajectory sample names</i>
-----------------	------------------------------------

---

**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	<i>GET user-defined landmarks</i>
---------------	-----------------------------------

---

**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	<i>Export trajectory graph</i>
----------------	--------------------------------

---

### 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

- \*Topic **datasets**
  - exSCE, [10](#)
- addTrail, [3](#)
- addTrail, SingleCellExperiment-method
  - (addTrail), [3](#)
- connectStates, [4](#)
- connectStates, SingleCellExperiment-method
  - (connectStates), [4](#)
- contrastTrailExpr, [6](#)
- contrastTrailExpr, SingleCellExperiment-method
  - (contrastTrailExpr), [6](#)
- embedSamples, [7](#)
- embedSamples, matrix-method
  - (embedSamples), [7](#)
- embedSamples, SingleCellExperiment-method
  - (embedSamples), [7](#)
- enrichment.test, [9](#)
- exSCE, [10](#)
- featureNames, SingleCellExperiment-method,
  - [11](#)
- filterTrajFeaturesByCOV, [12](#)
- filterTrajFeaturesByCOV, SingleCellExperiment-method
  - (filterTrajFeaturesByCOV), [12](#)
- filterTrajFeaturesByDL, [13](#)
- filterTrajFeaturesByDL, SingleCellExperiment-method
  - (filterTrajFeaturesByDL), [13](#)
- filterTrajFeaturesByFF, [14](#)
- filterTrajFeaturesByFF, SingleCellExperiment-method
  - (filterTrajFeaturesByFF), [14](#)
- findSpectrum, [15](#)
- findSpectrum, numeric-method
  - (findSpectrum), [15](#)
- findStates, [16](#)
- findStates, SingleCellExperiment-method
  - (findStates), [16](#)
- fitDynamic, [18](#)
- fitDynamic, SingleCellExperiment-method
  - (fitDynamic), [18](#)
- fitTrajectory, [19](#)
- fitTrajectory, SingleCellExperiment-method
  - (fitTrajectory), [19](#)
- landmarks, [20](#)
- landmarks, SingleCellExperiment-method
  - (landmarks), [20](#)
- latentSpace, [21](#)
- latentSpace, SingleCellExperiment-method
  - (latentSpace), [21](#)
- latentSpace<-, [22](#)
- latentSpace<-, SingleCellExperiment-method
  - (latentSpace<-), [22](#)
- manifold2D, [23](#)
- manifold2D, SingleCellExperiment-method
  - (manifold2D), [23](#)
- manifold2D<-, [23](#)
- manifold2D<-, SingleCellExperiment-method
  - (manifold2D<-), [23](#)
- pca, [24](#)
- pca, SingleCellExperiment-method (pca),
  - [24](#)
- phenoNames, [25](#)
- phenoNames, SingleCellExperiment-method
  - (phenoNames), [25](#)
- plotDynamic, [26](#)
- plotDynamic, SingleCellExperiment-method
  - (plotDynamic), [26](#)
- plotManifold, [27](#)
- plotManifold, SingleCellExperiment-method
  - (plotManifold), [27](#)
- plotMap, [28](#)
- plotMap, SingleCellExperiment-method
  - (plotMap), [28](#)
- plotStateExpression, [29](#)
- plotStateExpression, SingleCellExperiment-method
  - (plotStateExpression), [29](#)
- plotStateSize, [30](#)
- plotStateSize, SingleCellExperiment-method
  - (plotStateSize), [30](#)
- plotStateTrajectory, [31](#)
- plotStateTrajectory, SingleCellExperiment-method
  - (plotStateTrajectory), [31](#)
- plotTrail, [32](#)
- plotTrail, SingleCellExperiment-method
  - (plotTrail), [32](#)

plotTrajectoryFit, 33  
 plotTrajectoryFit, SingleCellExperiment-method  
     (plotTrajectoryFit), 33  
  
 read.ygraphml, 34  
 removeTrail, 35  
 removeTrail, SingleCellExperiment-method  
     (removeTrail), 35  
  
 sampleNames, SingleCellExperiment-method,  
     36  
 selectTrajectory, 37  
 selectTrajectory, SingleCellExperiment-method  
     (selectTrajectory), 37  
 showTrajInfo, 38  
 showTrajInfo, SingleCellExperiment-method  
     (showTrajInfo), 38  
 simulate\_exprs, 38  
 states, 39  
 states, SingleCellExperiment-method  
     (states), 39  
 states<-, 40  
 states<-, SingleCellExperiment-method  
     (states<-), 40  
 stateTrajLayout<-, 41  
 stateTrajLayout<-, SingleCellExperiment-method  
     (stateTrajLayout<-), 41  
  
 trailNames, 41  
 trailNames, SingleCellExperiment-method  
     (trailNames), 41  
 trailNames<-, 42  
 trailNames<-, SingleCellExperiment-method  
     (trailNames<-), 42  
 trails, 43  
 trails, SingleCellExperiment-method  
     (trails), 43  
 trajComponents, 44  
 trajComponents, SingleCellExperiment-method  
     (trajComponents), 44  
 trajFeatureNames, 44  
 trajFeatureNames, SingleCellExperiment-method  
     (trajFeatureNames), 44  
 trajFeatureNames<-, 45  
 trajFeatureNames<-, SingleCellExperiment-method  
     (trajFeatureNames<-), 45  
 trajLayout, 46  
 trajLayout, SingleCellExperiment-method  
     (trajLayout), 46  
 trajLayout<-, 46  
 trajLayout<-, SingleCellExperiment-method  
     (trajLayout<-), 46  
 trajResiduals, 47  
 trajResiduals, SingleCellExperiment-method  
     (trajResiduals), 47  
 trajSampleNames, 48  
 trajSampleNames, SingleCellExperiment-method  
     (trajSampleNames), 48  
  
 userLandmarks, 49  
 userLandmarks, SingleCellExperiment-method  
     (userLandmarks), 49  
 userLandmarks<-, 50  
 userLandmarks<-, SingleCellExperiment-method  
     (userLandmarks<-), 50  
 write.ygraphml, 51  
 write.ygraphml, SingleCellExperiment-method  
     (write.ygraphml), 51