Package ‘splatter’

March 23, 2017

Type Package

Title Simple Simulation of Single-cell RNA Sequencing Data

Version 0.99.12

Date 2017-03-22

Author Luke Zappia

Maintainer Luke Zappia <luke.zappia@mcri.edu.au>

Description Splatter is a package for the simulation of single-cell RNA sequencing count data. It provides a simple interface for creating complex simulations that are reproducible and well-documented.

License GPL-3 + file LICENSE

LazyData TRUE

Depends R (>= 3.4), scater

Imports fitdistrplus, edgeR, stats, locfit, akima, Biobase, checkmate, methods, utils, matrixStats, ggplot2, scales, BiocParallel

Suggests testthat, scran, progress, lme4, pscl, scDD, knitr, rmarkdown, BiocStyle, covr, S4Vectors, SummarizedExperiment

biocViews SingleCell, RNASeq, Transcriptomics, GeneExpression, Sequencing, Software

URL https://github.com/Oshlack/splatter

BugReports https://github.com/Oshlack/splatter/issues

RoxygenNote 6.0.1

VignetteBuilder knitr

NeedsCompilation no

R topics documented:

addFeatureStats .......................................................... 3
addGeneLengths ............................................................. 3
bridge ................................................................. 4
compareSCESets ........................................................... 5
diffSCESets ............................................................... 6
expandParams ............................................................ 7
getLNormFactors .......................................................... 8
R topics documented:

getParam .......................................................... 8
getParams .......................................................... 9
getPathOrder .......................................................... 9
listSims ............................................................. 10
logistic .............................................................. 10
lun2Estimate ........................................................ 11
Lun2Params .......................................................... 12
lun2Simulate .......................................................... 12
lunEstimate .......................................................... 13
LunParams ............................................................ 14
lunSimulate ............................................................ 15
newParams ............................................................ 16
Params ................................................................. 16
rbindMatched .......................................................... 17
scDDEstimate .......................................................... 17
SCDDParams ........................................................... 18
scDDSimulate ........................................................... 19
setParam .............................................................. 20
setParams ............................................................. 21
setParamsUnchecked .................................................... 21
setParamUnchecked ...................................................... 22
showPP ................................................................. 23
simpleEstimate ....................................................... 23
SimpleParams .......................................................... 24
simpleSimulate .......................................................... 24
splatEstBCV ............................................................ 25
splatEstDropout .......................................................... 26
splatEstimate ............................................................ 26
splatEstLib ............................................................... 27
splatEstMean ............................................................. 28
splatEstOutlier ........................................................... 28
SplatParams .............................................................. 29
splatSimBCVMeans ....................................................... 30
splatSimCellMeans ....................................................... 31
splatSimDE .............................................................. 31
splatSimDropout .......................................................... 32
splatSimGeneMeans ...................................................... 32
splatSimLibSizes ......................................................... 33
splatSimTrueCounts ...................................................... 33
splatSimulate ............................................................ 34
splatter ................................................................. 36
winsorize ............................................................... 36

Index ................................................................. 37
**addFeatureStats**

---

### Description

Add additional feature statistics to an SCESet object

### Usage

```r
addFeatureStats(sce, value = c("counts", "cpm", "tpm", "fpkm"), log = FALSE, offset = 1, no.zeros = FALSE)
```

### Arguments

- **sce**: SCESet to add feature statistics to.
- **value**: the expression value to calculate statistics for. Options are "counts", "cpm", "tpm" or "fpkm". The values need to exist in the given SCESet.
- **log**: logical. Whether to take log2 before calculating statistics.
- **offset**: offset to add to avoid taking log of zero.
- **no.zeros**: logical. Whether to remove all zeros from each feature before calculating statistics.

### Details

Currently adds the following statistics: mean, variance, coefficient of variation, median and median absolute deviation. Statistics are added to the `fData` slot and are named `Stat[Log]Value[No0]` where `Log` and `No0` are added if those arguments are true. UpperCamelCase is used to differentiate these columns from those added by `scater`.

### Value

SCESet with additional feature statistics

---

**addGeneLengths**

---

### Description

Add gene lengths to an SCESet object

### Usage

```r
addGeneLengths(sce, method = c("generate", "sample"), loc = 7.9, scale = 0.7, lengths = NULL)
```
Arguments

- **sce**: SCESet to add gene lengths to.
- **method**: Method to use for creating lengths.
- **loc**: Location parameter for the generate method.
- **scale**: Scale parameter for the generate method.
- **lengths**: Vector of lengths for the sample method.

Details

This function adds simulated gene lengths to the fData slot of an SCESet object that can be used for calculating length normalised expression values such as TPM or FPKM. The generate simulates lengths using a (rounded) log-normal distribution, with the default loc and scale parameters based on human coding genes. Alternatively the sample method can be used which randomly samples lengths (with replacement) from a supplied vector.

Value

SCESet with added gene lengths

Examples

```r
# Default generate method
sce <- simpleSimulate()
sce <- addGeneLengths(sce)
head(fData(sce))

# Sample method (human coding genes)
## Not run:
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
library(GenomicFeatures)

## Not run:
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
tax.lens <- transcriptLengths(txdb, with.cds_len = TRUE)
tax.lens <- tx.lens[tx.lens$cds_len > 0, ]
gene.lens <- max(splitAsList(tx.lens$tx_len, tx.lens$gene_id))
sce <- addGeneLengths(sce, method = "sample", lengths = gene.lens)
## End(Not run)
```

bridge

Brownian bridge

Description

Calculate a smoothed Brownian bridge between two points. A Brownian bridge is a random walk with fixed end points.

Usage

`bridge(x = 0, y = 0, N = 5, n = 100, sigma.fac = 0.8)`
**compareSCESets**  

**Arguments**

- **x**: starting value.
- **y**: end value.
- **N**: number of steps in random walk.
- **n**: number of points in smoothed bridge.
- **sigma.fact**: multiplier specifying how extreme each step can be.

**Value**

Vector of length $n$ following a path from $x$ to $y$.

---

**compareSCESets**

*Compare SCESet objects*

**Description**

Combine the data from several SCESet objects and produce some basic plots comparing them.

**Usage**

`compareSCESets(sces)`

**Arguments**

- **sces**: named list of SCESet objects to combine and compare.

**Details**

The returned list has three items:

- **FeatureData**: Combined feature data from the provided SCESets.
- **PhenoData**: Combined pheno data from the provided SCESets.

**Plots**

- **Means**: Boxplot of mean distribution.
- **Variances**: Boxplot of variance distribution.
- **MeanVar**: Scatter plot with fitted lines showing the mean-variance relationship.
- **LibrarySizes**: Boxplot of the library size distribution.
- **ZerosGene**: Boxplot of the percentage of each gene that is zero.
- **ZerosCell**: Boxplot of the percentage of each cell that is zero.
- **MeanZeros**: Scatter plot with fitted lines showing the mean-dropout relationship.

The plots returned by this function are created using `ggplot` and are only a sample of the kind of plots you might like to consider. The data used to create these plots is also returned and should be in the correct format to allow you to create further plots using `ggplot`.

**Value**

List containing the combined datasets and plots.
Examples

```r
sim1 <- splatSimulate(nGenes = 1000, groupCells = 20)
sim2 <- simpleSimulate(nGenes = 1000, nCells = 20)
comparison <- compareSCESets(list(Splat = sim1, Simple = sim2))
names(comparison)
names(comparison$Plots)
```

---

**diffSCESets**  
*Diff SCESet objects*

**Description**

Combine the data from several SCESet objects and produce some basic plots comparing them to a reference.

**Usage**

```r
diffSCESets(sces, ref)
```

**Arguments**

- `sces` named list of SCESet objects to combine and compare.
- `ref` string giving the name of the SCESet to use as the reference

**Details**

This function aims to look at the differences between a reference SCESet and one or more others. It requires each SCESet to have the same dimensions. Properties are compared by ranks, for example when comparing the means the values are ordered and the differences between the reference and another dataset plotted. A series of Q-Q plots are also returned.

The returned list has five items:

- **Reference** The SCESet used as the reference.
- **FeatureData** Combined feature data from the provided SCESets.
- **PhenoData** Combined pheno data from the provided SCESets.
- **Plots** Difference plots
  - **Means** Boxplot of mean differences.
  - **Variances** Boxplot of variance differences.
  - **MeanVar** Scatter plot showing the difference from the reference variance across expression ranks.
  - **LibrarySizes** Boxplot of the library size differences.
  - **ZerosGene** Boxplot of the differences in the percentage of each gene that is zero.
  - **ZerosCell** Boxplot of the differences in the percentage of each cell that is zero.
  - **MeanZeros** Scatter plot showing the difference from the reference percentage of zeros across expression ranks.
- **QQPlots** Quantile-Quantile plots
  - **Means** Q-Q plot of the means.
  - **Variances** Q-Q plot of the variances.
expandParams

LibrarySizes  Q-Q plot of the library sizes.
ZerosGene    Q-Q plot of the percentage of zeros per gene.
ZerosCell    Q-Q plot of the percentage of zeros per cell.

The plots returned by this function are created using ggplot and are only a sample of the kind of plots you might like to consider. The data used to create these plots is also returned and should be in the correct format to allow you to create further plots using ggplot.

Value

List containing the combined datasets and plots.

Examples

```r
sim1 <- splatSimulate(nGenes = 1000, groupCells = 20)
sim2 <- simpleSimulate(nGenes = 1000, nCells = 20)
difference <- diffSCESets(list(Splat = sim1, Simple = sim2), ref = "Simple")
names(difference)
names(difference$Plots)
```

---

**Description**

Expand the parameters that can be vectors so that they are the same length as the number of groups.

**Usage**

```r
expandParams(object, ...)
```

```r
## S4 method for signature 'LunParams'
expandParams(object)
```

```r
## S4 method for signature 'SplatParams'
expandParams(object)
```

**Arguments**

- `object`  object to expand.
- `...`     additional arguments.

**Value**

Expanded object.
getParam

Get log-normal factors

Description
Randomly generate multiplication factors from a log-normal distribution.

Usage
getLNormFactors(n.facs, sel.prob, neg.prob, fac.loc, fac.scale)

Arguments
- n.facs: Number of factors to generate.
- sel.prob: Probability that a factor will be selected to be different from 1.
- neg.prob: Probability that a selected factor is less than one.
- fac.loc: Location parameter for the log-normal distribution.
- fac.scale: Scale factor for the log-normal distribution.

Value
Vector containing generated factors.

getParam

Get a parameter

Description
Accessor function for getting parameter values.

Usage
getParam(object, name)

## S4 method for signature 'Params'
getParam(object, name)

Arguments
- object: object to get parameter from.
- name: name of the parameter to get.

Value
The extracted parameter value

Examples
params <- newSimpleParams()
getParam(params, "nGenes")
getParams

Description
Get multiple parameter values from a Params object.

Usage
getParams(params, names)

Arguments
params Params object to get values from.
names vector of names of the parameters to get.

Value
List with the values of the selected parameters.

Examples
params <- newSimpleParams()
getParams(params, c("nGenes", "nCells", "mean.rate"))

getPathOrder

Description
Identify the correct order to process paths so that preceding paths have already been simulated.

Usage
getPathOrder(path.from)

Arguments
path.from vector giving the path endpoints that each path orginates from.

Value
Vector giving the order to process paths in.
listSims  

**Description**

List all the simulations that are currently available in Splatter with a brief description.

**Usage**

```r
listSims(print = TRUE)
```

**Arguments**

- `print`  
  logical. Whether to print to the console.

**Value**

Invisibly returns a data.frame containing the information that is displayed.

**Examples**

```r
listSims()
```

logistic  

**Description**

Implementation of the logistic function

**Usage**

```r
logistic(x, x0, k)
```

**Arguments**

- `x`  
  value to apply the function to.
- `x0`  
  midpoint parameter. Gives the centre of the function.
- `k`  
  shape parameter. Gives the slope of the function.

**Value**

Value of logistic function with given parameters
lun2Estimate

Estimate Lun2 simulation parameters

Description

Estimate simulation parameters for the Lun2 simulation from a real dataset.

Usage

lun2Estimate(counts, plates, params = newLun2Params(), min.size = 200, verbose = TRUE, BPPARAM = SerialParam())

## S3 method for class 'SCESet'

lun2Estimate(counts, plates, params = newLun2Params(), min.size = 200, verbose = TRUE, BPPARAM = SerialParam())

## S3 method for class 'matrix'

lun2Estimate(counts, plates, params = newLun2Params(), min.size = 200, verbose = TRUE, BPPARAM = SerialParam())

Arguments

counts either a counts matrix or an SCESet object containing count data to estimate parameters from.

plates integer vector giving the plate that each cell originated from.

params Lun2Params object to store estimated values in.

min.size minimum size of clusters when identifying group of cells in the data.

verbose logical. Whether to show progress messages.

BPPARAM A BiocParallelParam instance giving the parallel back-end to be used. Default is SerialParam which uses a single core.

Details

See Lun2Params for more details on the parameters.

Value

LunParams object containing the estimated parameters.

Examples

## Not run:
data("sc_example_counts")
data("sc_example_cell_info")
plates <- factor(sc_example_cell_info$Mutation_Status)
params <- lun2Estimate(sc_example_counts, plates, min.size = 20)
params

## End(Not run)
Lun2Params

The Lun2Params class

Description

S4 class that holds parameters for the Lun simulation.

Parameters

The Lun2 simulation uses the following parameters:

- nGenes: The number of genes to simulate.
- nCells: The number of cells to simulate.
- [seed]: Seed to use for generating random numbers.
- [nPlates]: The number of plates to simulate.

**Plate parameters**

- plate.ingroup: Character vector giving the plates considered to be part of the “ingroup”.
- plate.mod: Plate effect modifier factor. The plate effect variance is divided by this value.
- plate.var: Plate effect variance.

**Gene parameters**

- gene.means: Mean expression for each gene.
- gene.disps: Dispersion for each gene.

**Cell parameters**

- cell.plates: Factor giving the plate that each cell comes from.
- cell.libSizes: Library size for each cell.
- cell.libMod: Modifier factor for library sizes. The library sizes are multiplied by this value.

**Differential expression parameters**

- de.nGenes: Number of differentially expressed genes.
- de.fc: Fold change for differentially expressed genes.

The parameters not shown in brackets can be estimated from real data using `lun2Estimate`. For details of the Lun2 simulation see `lun2Simulate`.

lun2Simulate

Lun2 simulation

Description

Simulate single-cell RNA-seq count data using the method described in Lun and Marioni "Overcoming confounding plate effects in differential expression analyses of single-cell RNA-seq data".

Usage

`lun2Simulate(params = newLun2Params(), zinb = FALSE, verbose = TRUE, ...)"
lunEstimate

Arguments

params Lun2Params object containing simulation parameters.
zinb logical. Whether to use a zero-inflated model.
verbose logical. Whether to print progress messages
... any additional parameter settings to override what is provided in params.

Details

The Lun2 simulation uses a negative-binomial distribution where the means and dispersions have been sampled from a real dataset (using lun2Estimate). The other core feature of the Lun2 simulation is the addition of plate effects. Differential expression can be added between two groups of plates (an “ingroup” and all other plates). Library size factors are also applied and optionally a zero-inflated negative-binomial can be used.

Value

SCESet containing simulated counts.

References

Paper: dx.doi.org/10.1101/073973
Code: https://github.com/MarioniLab/PlateEffects2016

Examples

sim <- lun2Simulate()

Description

Estimate simulation parameters for the Lun simulation from a real dataset.

Usage

lunEstimate(counts, params = newLunParams())

## S3 method for class 'SCESet'
lunEstimate(counts, params = newLunParams())

## S3 method for class 'matrix'
lunEstimate(counts, params = newLunParams())

Arguments

counts either a counts matrix or an SCESet object containing count data to estimate parameters from.
params LunParams object to store estimated values in.
Details

The nGenes and nCells parameters are taken from the size of the input data. No other parameters are estimated. See LunParams for more details on the parameters.

Value

LunParams object containing the estimated parameters.

Examples

data("sc_example_counts")
params <- lunEstimate(sc_example_counts)
params

---

LunParams  The LunParams class

Description

S4 class that holds parameters for the Lun simulation.

Parameters

The Lun simulation uses the following parameters:

- nGenes  The number of genes to simulate.
- nCells  The number of cells to simulate.
- nGroups] The number of groups to simulate.
- [groupCells] Vector giving the number of cells in each simulation group/path.
- [seed] Seed to use for generating random numbers.

Mean parameters  [mean.shape] Shape parameter for the mean gamma distribution.
- [mean.rate] Rate parameter for the mean gamma distribution.

Counts parameters  [count.disp] The dispersion parameter for the counts negative binomial distribution.

Differential expression parameters  [de.nGenes] The number of genes that are differentially expressed in each group
- [de.upProp] The proportion of differentially expressed genes that are up-regulated in each group
- [de.upFC] The fold change for up-regulated genes
- [de.downFC] The fold change for down-regulated genes

The parameters not shown in brackets can be estimated from real data using lunEstimate. For details of the Lun simulation see lunSimulate.
Simulate single-cell RNA-seq count data using the method described in Lun, Bach and Marioni “Pooling across cells to normalize single-cell RNA sequencing data with many zero counts”.

Usage

lunSimulate(params = newLunParams(), verbose = TRUE, ...)

Arguments

params LunParams object containing Lun simulation parameters.
verbose logical. Whether to print progress messages.
... any additional parameter settings to override what is provided in params.

Details

The Lun simulation generates gene mean expression levels from a gamma distribution with shape = mean.shape and rate = mean.rate. Counts are then simulated from a negative binomial distribution with mu = means and size = 1 / bcv.common. In addition each cell is given a size factor (2 ^ rnorm(nCells, mean = 0, sd = 0.5)) and differential expression can be simulated with fixed fold changes.

See LunParams for details of the parameters.

Value

SCESet object containing the simulated counts and intermediate values.

References

Lun ATL, Bach K, Marioni JC. Pooling across cells to normalize single-cell RNA sequencing data with many zero counts. Genome Biology (2016).


Code: https://github.com/MarioniLab/Deconvolution2016

Examples

sim <- lunSimulate()
Description

Create a new Params object. Functions exist for each of the different Params subtypes.

Usage

newLun2Params(...)
newLunParams(...)
newSCDDParams(...)
newSimpleParams(...)
newSplatParams(...)

Arguments

... additional parameters passed to setParams.

Value

New Params object.

Examples

params <- newSimpleParams()
params <- newSimpleParams(nGenes = 200, nCells = 10)

Params

The Params virtual class

Description

Virtual S4 class that all other Params classes inherit from.

Parameters

The Params class defines the following parameters:

- [nGenes] The number of genes to simulate.
- [nCells] The number of cells to simulate.
- seed Seed to use for generating random numbers.

The parameters shown in brackets can be estimated from real data.
**rbindMatched**

**Bind rows (matched)**

**Description**

Bind the rows of two data frames, keeping only the columns that are common to both.

**Usage**

`rbindMatched(df1, df2)`

**Arguments**

- `df1`: first data.frame to bind.
- `df2`: second data.frame to bind.

**Value**

data.frame containing rows from `df1` and `df2` but only common columns.

---

**scDDEstimate**

**Estimate scDD simulation parameters**

**Description**

Estimate simulation parameters for the scDD simulation from a real dataset.

**Usage**

`scDDEstimate(counts, conditions, params = newSCDDParams())`

```
## S3 method for class 'SCESet'
scDDEstimate(counts, conditions, params = newSCDDParams())
```

```
## S3 method for class 'matrix'
scDDEstimate(counts, conditions, params = newSCDDParams())
```

**Arguments**

- `counts`: either a counts matrix or an SCESet object containing count data to estimate parameters from.
- `conditions`: Vector giving the condition that each cell belongs to. Conditions can be 1 or 2.
- `params`: SCDDParams object to store estimated values in.

**Details**

This function is just a wrapper around `preprocess` that takes the output and converts it to a SCDDParams object. See `preprocess` for details.
SCDDParams

Value

SCDDParams object containing the estimated parameters.

Examples

data("sc_example_counts")
conditions <- sample(1:2, ncol(sc_example_counts), replace = TRUE)
params <- scDDEstimate(sc_example_counts, conditions)
params

SCDDParams

The SCDDParams class

Description

S4 class that holds parameters for the scDD simulation.

Parameters

The SCDD simulation uses the following parameters:

[nGenes] The number of genes to simulate (not used).
nCells The number of cells to simulate in each condition.
[seed] Seed to use for generating random numbers.
SCdat SummarizedExperiment containing real data.
[nDE] Number of DE genes to simulate.
[nDP] Number of DP genes to simulate.
[nDM] Number of DM genes to simulate.
[nDB] Number of DB genes to simulate.
[nEE] Number of EE genes to simulate.
[sd.range] Interval for fold change standard deviations.
[modeFC] Values for DP, DM and DB mode fold changes.
[varInflation] Variance inflation factors for each condition. If all equal to 1 will be set to NULL (default)
[condition] String giving the column that represents biological group of interest

The parameters not shown in brackets can be estimated from real data using scDDEstimate. See simulateSet for more details of the parameters. For details of the Splatter implementation of the scDD simulation see scDDSimulate.
scDDSimulate

**Description**

Simulate counts using the scDD method.

**Usage**

```r
scDDSimulate(params = newSCDDParams(), plots = FALSE, plot.file = NULL,
verbose = TRUE, ...)
```

**Arguments**

- `params`: SCDDParams object containing simulation parameters.
- `plots`: logical. whether to generate scDD fold change and validation plots.
- `plot.file`: File path to save plots as PDF.
- `verbose`: logical. Whether to print progress messages
- `...`: any additional parameter settings to override what is provided in `params`.

**Details**

This function is just a wrapper around `simulateSet` that takes a `SCDDParams`, runs the simulation then converts the output to an `SCESet` object. See `simulateSet` for more details of how the simulation works.

**Value**

SCESet containing simulated counts

**References**


Code: [https://github.com/kdkorthauer/scDD](https://github.com/kdkorthauer/scDD)

**Examples**

```r
## Not run:
sim <- scDDSimulate()
## End(Not run)
```
setParam

Description

Function for setting parameter values.

Usage

setParam(object, name, value)

## S4 method for signature 'Lun2Params'
setParam(object, name, value)

## S4 method for signature 'LunParams'
setParam(object, name, value)

## S4 method for signature 'Params'
setParam(object, name, value)

## S4 method for signature 'SCDDParams'
setParam(object, name, value)

## S4 method for signature 'SplatParams'
setParam(object, name, value)

Arguments

object  object to set parameter in.
name    name of the parameter to set.
value   value to set the parameter to.

Value

Object with new parameter value.

Examples

params <- newSimpleParams()
setParam(params, "nGenes", 100)
setParams

Description

Set multiple parameters in a Params object.

Usage

setParams(params, update = NULL, ...)

Arguments

params Params object to set parameters in.
update list of parameters to set where names(update) are the names of the parameters to set and the items in the list are values.
... additional parameters to set. These are combined with any parameters specified in update.

Details

Each parameter is set by a call to setParam. If the same parameter is specified multiple times it will be set multiple times. Parameters can be specified using a list via update (useful when collecting parameter values in some way) or individually (useful when setting them manually), see examples.

Value

Params object with updated values.

Examples

params <- newSimpleParams()
params
# Set individually
params <- setParams(params, nGenes = 1000, nCells = 50)
params
# Set via update list
params <- setParams(params, list(mean.rate = 0.2, mean.shape = 0.8))
params

setParamsUnchecked

Description

Set multiple parameters in a Params object.

Usage

setParamsUnchecked(params, update = NULL, ...)

**setParamUnchecked**

### Arguments

- **params**: Params object to set parameters in.
- **update**: list of parameters to set where `names(update)` are the names of the parameters to set and the items in the list are values.
- ... additional parameters to set. These are combined with any parameters specified in `update`.

### Details

Each parameter is set by a call to `setParam`. If the same parameter is specified multiple times it will be set multiple times. Parameters can be specified using a list via `update` (useful when collecting parameter values in some way) or individually (useful when setting them manually), see examples.

THE FINAL OBJECT IS NOT CHECKED FOR VALIDITY!

### Value

Params object with updated values.

---

**setParamUnchecked**  
*Set a parameter UNCHECKED*

### Description

Function for setting parameter values. THE OUTPUT IS NOT CHECKED FOR VALIDITY!

### Usage

```
setParamUnchecked(object, name, value)
```

### Arguments

- **object**: object to set parameter in.
- **name**: name of the parameter to set.
- **value**: value to set the parameter to.

### Value

Object with new parameter value.
**showPP**

*Show pretty print*

**Description**

Function used for pretty printing params object.

**Usage**

```r
showPP(params, pp)
```

**Arguments**

- `params` object to show.
- `pp` list specifying how the object should be displayed.

**Value**

Print params object to console

**simpleEstimate**

*Estimate simple simulation parameters*

**Description**

Estimate simulation parameters for the simple simulation from a real dataset.

**Usage**

```r
simpleEstimate(counts, params = newSimpleParams())
```

**Arguments**

- `counts` either a counts matrix or an SCESet object containing count data to estimate parameters from.
- `params` SimpleParams object to store estimated values in.

**Details**

The `nGenes` and `nCells` parameters are taken from the size of the input data. The mean parameters are estimated by fitting a gamma distribution to the library size normalised mean expression level using `fitdist`. See `SimpleParams` for more details on the parameters.
simpleSimulate

Value

SimpleParams object containing the estimated parameters.

Examples

data("sc_example_counts")
params <- simpleEstimate(sc_example_counts)
params

SimpleParams  The SimpleParams class

Description

S4 class that holds parameters for the simple simulation.

Parameters

The simple simulation uses the following parameters:

- nGenes  The number of genes to simulate.
- nCells  The number of cells to simulate.
- [seed]  Seed to use for generating random numbers.
- mean.shape  The shape parameter for the mean gamma distribution.
- mean.rate  The rate parameter for the mean gamma distribution.
- [count.disp]  The dispersion parameter for the counts negative binomial distribution.

The parameters not shown in brackets can be estimated from real data using simpleEstimate. For details of the simple simulation see simpleSimulate.

simpleSimulate  Simple simulation

Description

Simulate counts from a simple negative binomial distribution without simulated library sizes, differential expression etc.

Usage

simpleSimulate(params = newSimpleParams(), verbose = TRUE, ...)

Arguments

params  SimpleParams object containing simulation parameters.
verbose  logical. Whether to print progress messages
...  any additional parameter settings to override what is provided in params.
**Details**

Gene means are simulated from a gamma distribution with \( \text{shape} = \text{mean.shape} \) and \( \text{rate} = \text{mean.rate} \). Counts are then simulated from a negative binomial distribution with \( \mu = \text{means} \) and \( \text{size} = \frac{1}{\text{counts.disp}} \). See SimpleParams for more details of the parameters.

**Value**

SCESet containing simulated counts

**Examples**

```r
sim <- simpleSimulate()
# Override default parameters
sim <- simpleSimulate(nGenes = 1000, nCells = 50)
```

---

**splatEstBCV**

*Estimate Splat Biological Coefficient of Variation parameters*

**Description**

Parameters are estimated using the `estimateDisp` function in the edgeR package.

**Usage**

`splatEstBCV(counts, params)`

**Arguments**

- `counts`: counts matrix to estimate parameters from.
- `params`: SplatParams object to store estimated values in.

**Details**

The `estimateDisp` function is used to estimate the common dispersion and prior degrees of freedom. See `estimateDisp` for details. When estimating parameters on simulated data we found a broadly linear relationship between the true underlying common dispersion and the edgeR estimate, therefore we apply a small correction, \( \text{disp} = 0.1 + 0.25 \times \text{edgeR.disp} \).

**Value**

SplatParams object with estimated values.
splatEstDropout

*Estimate Splat dropout parameters*

**Description**

Estimate the midpoint and shape parameters for the logistic function used when simulating dropout. Also estimates whether dropout is likely to be present in the dataset.

**Usage**

`splatEstDropout(norm.counts, params)`

**Arguments**

- `norm.counts`: library size normalised counts matrix.
- `params`: SplatParams object to store estimated values in.

**Details**

Logistic function parameters are estimated by fitting a logistic function to the relationship between log2 mean gene expression and the proportion of zeros in each gene. See `nls` for details of fitting. The presence of dropout is determined by comparing the observed number of zeros in each gene to the expected number of zeros from a negative binomial distribution with the gene mean and a dispersion of 0.1. If the maximum difference between the observed number of zeros and the expected number is greater than 10 percent of the number of cells `(max(obs.zeros - exp.zeros) > 0.1 * ncol(norm.counts))`, then dropout is considered to be present in the dataset. This is a somewhat crude measure but should give a reasonable indication. A more accurate approach is to look at a plot of log2 mean expression vs the difference between observed and expected number of zeros across all genes.

**Value**

SplatParams object with estimated values.

---

splatEstimate

*Estimate Splat simulation parameters*

**Description**

Estimate simulation parameters for the Splat simulation from a real dataset. See the individual estimation functions for more details on how this is done.

**Usage**

`splatEstimate(counts, params = newSplatParams())`

## S3 method for class 'SCESet'
`splatEstimate(counts, params = newSplatParams())`

## S3 method for class 'matrix'
`splatEstimate(counts, params = newSplatParams())`
**splatEstLib**

**Arguments**

- `counts`: either a counts matrix or an SCESet object containing count data to estimate parameters from.
- `params`: SplatParams object to store estimated values in.

**Value**

SplatParams object containing the estimated parameters.

**See Also**

- `splatEstMean`
- `splatEstLib`
- `splatEstOutlier`
- `splatEstBCV`
- `splatEstDropout`

**Examples**

```r
data("sc_example_counts")
params <- splatEstimate(sc_example_counts)
params
```

---

### splatEstLib

*Estimate Splat library size parameters*

**Description**

A log-normal distribution is fitted to the library sizes and the estimated parameters are added to the params object. See `fitdist` for details on the fitting.

**Usage**

```r
splatEstLib(counts, params)
```

**Arguments**

- `counts`: counts matrix to estimate parameters from.
- `params`: SplatParams object to store estimated values in.

**Value**

SplatParams object with estimated values.
splatEstMean  Estimate Splat mean parameters

Description

Estimate rate and shape parameters for the gamma distribution used to simulate gene expression means.

Usage

splatEstMean(norm.counts, params)

Arguments

- `norm.counts`: library size normalised counts matrix.
- `params`: SplatParams object to store estimated values in.

Details

Parameter for the gamma distribution are estimated by fitting the mean normalised counts using `fitdist`. The 'maximum goodness-of-fit estimation' method is used to minimise the Cramer-von Mises distance. This can fail in some situations, in which case the 'method of moments estimation' method is used instead. Prior to fitting the means are winsorized by setting the top and bottom 10 percent of values to the 10th and 90th percentiles.

Value

SplatParams object with estimated values.

splatEstOutlier  Estimate Splat expression outlier parameters

Description

Parameters are estimated by comparing means of individual genes to the median mean expression level.

Usage

splatEstOutlier(norm.counts, params)

Arguments

- `norm.counts`: library size normalised counts matrix.
- `params`: SplatParams object to store estimated values in.
Details

Expression outlier genes are detected using the Median Absolute Deviation (MAD) from median method. If the log2 mean expression of a gene is greater than two MADs above the median log2 mean expression it is designated as an outlier. The proportion of outlier genes is used to estimate the outlier probability. Factors for each outlier gene are calculated by dividing mean expression by the median mean expression. A log-normal distribution is then fitted to these factors in order to estimate the outlier factor location and scale parameters using `fitdist`.

Value

SplatParams object with estimated values.

---

| SplatParams | The SplatParams class |

Description

S4 class that holds parameters for the Splatter simulation.

Parameters

The Splatter simulation requires the following parameters:

- **nGenes** The number of genes to simulate.
- **nCells** The number of cells to simulate.
- **[nGroups]** The number of groups or paths to simulate.
- **[groupCells]** Vector giving the number of cells in each simulation group/path.
- **[seed]** Seed to use for generating random numbers.

Mean parameters

- **mean.shape** Shape parameter for the mean gamma distribution.
- **mean.rate** Rate parameter for the mean gamma distribution.

Library size parameters

- **lib.loc** Location (meanlog) parameter for the library size log-normal distribution.
- **lib.scale** Scale (sdlog) parameter for the library size log-normal distribution.

Expression outlier parameters

- **out.prob** Probability that a gene is an expression outlier.
- **out.facLoc** Location (meanlog) parameter for the expression outlier factor log-normal distribution.
- **out.facScale** Scale (sdlog) parameter for the expression outlier factor log-normal distribution.

Differential expression parameters

- **[de.prob]** Probability that a gene is differentially expressed in a group. Can be a vector.
- **[de.loProb]** Probability that a differentially expressed gene is down-regulated. Can be a vector.
- **[de.facLoc]** Location (meanlog) parameter for the differential expression factor log-normal distribution. Can be a vector.
- **[de.facScale]** Scale (sdlog) parameter for the differential expression factor log-normal distribution. Can be a vector.
**Biological Coefficient of Variation parameters**  
bcv.common  Underlying common dispersion across all genes.

bcv.df  Degrees of Freedom for the BCV inverse chi-squared distribution.

**Dropout parameters**  
dropout.present  Logical. Whether to simulate dropout.

dropout.mid  Midpoint parameter for the dropout logistic function.

**Differentiation path parameters**  
[path.from]  Vector giving the originating point of each path. This allows path structure such as a cell type which differentiates into an intermediate cell type that then differentiates into two mature cell types. A path structure of this form would have a "from" parameter of c(0, 1, 1) (where 0 is the origin). If no vector is given all paths will start at the origin.

[path.length]  Vector giving the number of steps to simulate along each path. If a single value is given it will be applied to all paths.

[path.skew]  Vector giving the skew of each path. Values closer to 1 will give more cells towards the starting population, values closer to 0 will give more cells towards the final population. If a single value is given it will be applied to all paths.

[path.nonlinearProb]  Probability that a gene follows a non-linear path along the differentiation path. This allows more complex gene patterns such as a gene being equally expressed at the beginning an end of a path but lowly expressed in the middle.

[path.sigmaFac]  Sigma factor for non-linear gene paths. A higher value will result in more extreme non-linear variations along a path.

The parameters not shown in brackets can be estimated from real data using `splatEstimate`. For details of the Splatter simulation see `splatSimulate`.

---

**splatSimBCVMeans**  
*Simulate BCV means*

**Description**  
Simulate means for each gene in each cell that are adjusted to follow a mean-variance trend using Biological Coefficient of Variation taken from an inverse gamma distribution.

**Usage**  
splatSimBCVMeans(sim, params)

**Arguments**  
sim  SCESet to add BCV means to.

params  SplatParams object with simulation parameters.

**Value**  
SCESet with simulated BCV means.
**splatSimCellMeans**

Simulate a gene by cell matrix giving the mean expression for each gene in each cell. Cells start with the mean expression for the group they belong to (when simulating groups) or cells are assigned the mean expression from a random position on the appropriate path (when simulating paths). The selected means are adjusted for each cell’s expected library size.

**Usage**

splatSimSingleCellMeans(sim, params)
splatSimGroupCellMeans(sim, params)
splatSimPathCellMeans(sim, params)

**Arguments**

- **sim** SCESet to add cell means to.
- **params** SplatParams object with simulation parameters.

**Value**

SCESet with added cell means.

**splatSimDE**

Simulate group differential expression

Simulate differential expression. Differential expression factors for each group are produced using getLNormFactors and these are added along with updated means for each group. For paths care is taken to make sure they are simulated in the correct order.

**Usage**

splatSimGroupDE(sim, params)
splatSimPathDE(sim, params)

**Arguments**

- **sim** SCESet to add differential expression to.
- **params** SplatParams object with simulation parameters.

**Value**

SCESet with simulated differential expression.
splatSimDropout  

**Simulate dropout**

**Description**

A logistic function is used to form a relationship between the expression level of a gene and the probability of dropout, giving a probability for each gene in each cell. These probabilities are used in a Bernoulli distribution to decide which counts should be dropped.

**Usage**

```
splatSimDropout(sim, params)
```

**Arguments**

- `sim`: SCESet to add dropout to.
- `params`: SplatParams object with simulation parameters.

**Value**

SCESet with simulated dropout and observed counts.

---

splatSimGeneMeans  

**Simulate gene means**

**Description**

Simulate gene means from a gamma distribution. Also simulates outlier expression factors. Genes with an outlier factor not equal to 1 are replaced with the median mean expression multiplied by the outlier factor.

**Usage**

```
splatSimGeneMeans(sim, params)
```

**Arguments**

- `sim`: SCESet to add gene means to.
- `params`: SplatParams object with simulation parameters.

**Value**

SCESet with simulated gene means.
**splatSimLibSizes**    
*Simulate library sizes*

**Description**
Simulate expected library sizes from a log-normal distribution

**Usage**
splatSimLibSizes(sim, params)

**Arguments**
- **sim**: SCESet to add library size to.
- **params**: SplatParams object with simulation parameters.

**Value**
SCESet with simulated library sizes.

**splatSimTrueCounts**    
*Simulate true counts*

**Description**
Simulate a true counts matrix. Counts are simulated from a poisson distribution where each gene in each cell has its own mean based on the group (or path position), expected library size and BCV.

**Usage**
splatSimTrueCounts(sim, params)

**Arguments**
- **sim**: SCESet to add true counts to.
- **params**: SplatParams object with simulation parameters.

**Value**
SCESet with simulated true counts.
splatSimulate  

**Description**
Simulate count data from a fictional single-cell RNA-seq experiment using the Splat method.

**Usage**

```r
splatSimulate(params = newSplatParams(), method = c("single", "groups", "paths"), verbose = TRUE, ...)
```

```r
splatSimulateSingle(params = newSplatParams(), verbose = TRUE, ...)
```

```r
splatSimulateGroups(params = newSplatParams(), verbose = TRUE, ...)
```

```r
splatSimulatePaths(params = newSplatParams(), verbose = TRUE, ...)
```

**Arguments**

- `params`  
  SplatParams object containing parameters for the simulation. See SplatParams for details.

- `method`  
  which simulation method to use. Options are "single" which produces a single population, "groups" which produces distinct groups (eg. cell types) or "paths" which selects cells from continuous trajectories (eg. differentiation processes).

- `verbose`  
  logical. Whether to print progress messages.

- `...`  
  any additional parameter settings to override what is provided in params.

**Details**
Parameters can be set in a variety of ways. If no parameters are provided the default parameters are used. Any parameters in params can be overridden by supplying additional arguments through a call to setParams. This design allows the user flexibility in how they supply parameters and allows small adjustments without creating a new SplatParams object. See examples for a demonstration of how this can be used.

The simulation involves the following steps:
1. Set up simulation object
2. Simulate library sizes
3. Simulate gene means
4. Simulate groups/paths
5. Simulate BCV adjusted cell means
6. Simulate true counts
7. Simulate dropout
8. Create final SCESet object

The final output is an SCESet object that contains the simulated counts but also the values for various intermediate steps. These are stored in the phenoData (for cell specific information), featureData (for gene specific information) or assayData (for gene by cell matrices) slots. This additional information includes:
phenodata | Cell | Unique cell identifier. |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Group</td>
<td>The group or path the cell belongs to.</td>
<td></td>
</tr>
<tr>
<td>ExpLibSize</td>
<td>The expected library size for that cell.</td>
<td></td>
</tr>
<tr>
<td>Step (paths only)</td>
<td>how far along the path each cell is.</td>
<td></td>
</tr>
</tbody>
</table>

featureData | Gene | Unique gene identifier. |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseGeneMean</td>
<td>The base expression level for that gene.</td>
<td></td>
</tr>
<tr>
<td>OutlierFactor</td>
<td>Expression outlier factor for that gene. Values of 1 indicate the gene is not an expression outlier.</td>
<td></td>
</tr>
<tr>
<td>GeneMean</td>
<td>Expression level after applying outlier factors.</td>
<td></td>
</tr>
<tr>
<td>DEFac</td>
<td>The differential expression factor for each gene in a particular group. Values of 1 indicate the gene is not differentially expressed.</td>
<td></td>
</tr>
<tr>
<td>GeneMean[Group]</td>
<td>Expression level of a gene in a particular group after applying differential expression factors.</td>
<td></td>
</tr>
<tr>
<td>SigmaFac[Path]</td>
<td>Factor applied to genes that have non-linear changes in expression along a path.</td>
<td></td>
</tr>
</tbody>
</table>

assayData | BaseCellMeans | The expression of genes in each cell adjusted for expected library size. |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BCV</td>
<td>The Biological Coefficient of Variation for each gene in each cell.</td>
<td></td>
</tr>
<tr>
<td>CellMeans</td>
<td>The expression level of genes in each cell adjusted for BCV.</td>
<td></td>
</tr>
<tr>
<td>TrueCounts</td>
<td>The simulated counts before dropout.</td>
<td></td>
</tr>
<tr>
<td>Dropout</td>
<td>Logical matrix showing which values have been dropped in which cells.</td>
<td></td>
</tr>
</tbody>
</table>

Values that have been added by Splatter are named using CamelCase in order to differentiate them from the values added by Scater which uses underscore_naming.

Value

SCESet object containing the simulated counts and intermediate values.

See Also

`splatSimLibSizes`, `splatSimGeneMeans`, `splatSimDE`, `splatSimCellMeans`, `splatSimBCVMeans`, `splatSimTrueCounts`, `splatSimDropout`

Examples

```r
# Simulation with default parameters
## Not run:
sim <- splatSimulate()
# Simulation with different number of genes
sim <- splatSimulate(nGenes = 1000)
# Simulation with custom parameters
params <- newSplatParams(nGenes = 100, mean.rate = 0.5)
sim <- splatSimulate(params)
# Simulation with adjusted custom parameters
sim <- splatSimulate(params, mean.rate = 0.6, out.prob = 0.2)
# Simulate groups
sim <- splatSimulate(method = "groups")
# Simulate paths
sim <- splatSimulate(method = "paths")
## End(Not run)
```
Description

splatter is a package for the well-documented and reproducible simulation of single-cell RNA-seq count data.

Details

As well as its own simulation model splatter provides functions for the estimation of model parameters.

Description

Set outliers in a numeric vector to a specified percentile.

Usage

winsorize(x, q)

Arguments

x Numeric vector to winsorize
q Percentile to set from each end

Value

Winsorized numeric vector
## Index

<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>addFeatureStats</td>
<td>3</td>
</tr>
<tr>
<td>addGeneLengths</td>
<td>3</td>
</tr>
<tr>
<td>assayData</td>
<td>34</td>
</tr>
<tr>
<td>BiocParallelParam</td>
<td>11</td>
</tr>
<tr>
<td>bridge</td>
<td>4</td>
</tr>
<tr>
<td>compareSCESets</td>
<td>5</td>
</tr>
<tr>
<td>diffSCESets</td>
<td>6</td>
</tr>
<tr>
<td>estimateDisp</td>
<td>25</td>
</tr>
<tr>
<td>expandParams, LunParams-method</td>
<td>7</td>
</tr>
<tr>
<td>expandParams, SplatParams-method</td>
<td>7</td>
</tr>
<tr>
<td>featureData</td>
<td>34</td>
</tr>
<tr>
<td>fitdist</td>
<td>23, 27–29</td>
</tr>
<tr>
<td>getLNormFactors</td>
<td>8, 31</td>
</tr>
<tr>
<td>getParam, Params-method (getParam)</td>
<td>8</td>
</tr>
<tr>
<td>getParams</td>
<td>9</td>
</tr>
<tr>
<td>getPathOrder</td>
<td>9</td>
</tr>
<tr>
<td>ggplot</td>
<td>5, 7</td>
</tr>
<tr>
<td>listSims</td>
<td>10</td>
</tr>
<tr>
<td>logistic</td>
<td>10</td>
</tr>
<tr>
<td>lun2Estimate, Lun2Params</td>
<td>11, 12, 13</td>
</tr>
<tr>
<td>Lun2Params, SCDDParams</td>
<td>18, 19</td>
</tr>
<tr>
<td>lun2Simulate</td>
<td>12, 12</td>
</tr>
<tr>
<td>lunEstimate</td>
<td>13, 14</td>
</tr>
<tr>
<td>LunParams, SerialParam</td>
<td>11</td>
</tr>
<tr>
<td>LunParams, SimpleParams</td>
<td>11</td>
</tr>
<tr>
<td>LunParams, SplatParams</td>
<td>29, 34</td>
</tr>
<tr>
<td>newLun2Params (newParams)</td>
<td>16</td>
</tr>
<tr>
<td>newLunParams (newParams)</td>
<td>16</td>
</tr>
<tr>
<td>newParams</td>
<td>16</td>
</tr>
<tr>
<td>newSCDDParams (newParams)</td>
<td>16</td>
</tr>
<tr>
<td>newSimpleParams (newParams)</td>
<td>16</td>
</tr>
<tr>
<td>newSplatParams (newParams)</td>
<td>16</td>
</tr>
<tr>
<td>nls</td>
<td>26</td>
</tr>
<tr>
<td>Params</td>
<td>16</td>
</tr>
<tr>
<td>Params-class (Params)</td>
<td>16</td>
</tr>
<tr>
<td>phenoData</td>
<td>34</td>
</tr>
<tr>
<td>preprocess</td>
<td>17</td>
</tr>
<tr>
<td>rbindMatched</td>
<td>17</td>
</tr>
<tr>
<td>scDDEstimate</td>
<td>17, 18</td>
</tr>
<tr>
<td>SCDDParams, SCDDParams</td>
<td>18, 19</td>
</tr>
<tr>
<td>SCDDSimulate</td>
<td>18, 19</td>
</tr>
<tr>
<td>SCEESet</td>
<td>19, 34</td>
</tr>
<tr>
<td>SerialParam</td>
<td>11</td>
</tr>
<tr>
<td>setParam, Lun2Params-method (setParam)</td>
<td>20</td>
</tr>
<tr>
<td>setParam, SCDDParams-method (setParam)</td>
<td>20</td>
</tr>
<tr>
<td>setParam, SplatParams-method (setParam)</td>
<td>20</td>
</tr>
<tr>
<td>setParams</td>
<td>16, 21, 34</td>
</tr>
<tr>
<td>setParamsUnchecked</td>
<td>21</td>
</tr>
<tr>
<td>setParamUnchecked</td>
<td>22</td>
</tr>
<tr>
<td>setParamUnchecked, Params-method</td>
<td>22</td>
</tr>
<tr>
<td>showPP</td>
<td>23</td>
</tr>
<tr>
<td>simpleEstimate, SimpleParams</td>
<td>23, 24</td>
</tr>
<tr>
<td>SimpleParams-class (SimpleParams)</td>
<td>24</td>
</tr>
<tr>
<td>simulateSet</td>
<td>18, 19</td>
</tr>
<tr>
<td>splatEstBCV</td>
<td>25, 27</td>
</tr>
<tr>
<td>splatEstDropout</td>
<td>26, 27</td>
</tr>
<tr>
<td>splatEstimate</td>
<td>26, 30</td>
</tr>
<tr>
<td>splatEstLib</td>
<td>27, 27</td>
</tr>
<tr>
<td>splatEstMean</td>
<td>27, 28</td>
</tr>
<tr>
<td>splatOutlier</td>
<td>27, 28</td>
</tr>
<tr>
<td>SplatParams</td>
<td>29, 34</td>
</tr>
<tr>
<td>SplatParams-class (SplatParams)</td>
<td>29</td>
</tr>
<tr>
<td>splatSimBCVMeans</td>
<td>30, 35</td>
</tr>
</tbody>
</table>
splatSimCellMeans, 31, 35
splatSimDE, 31, 35
splatSimDropout, 32, 35
splatSimGeneMeans, 32, 35
splatSimGroupCellMeans
   (splatSimCellMeans), 31
splatSimGroupDE (splatSimDE), 31
splatSimLibSizes, 33, 35
splatSimPathCellMeans
   (splatSimCellMeans), 31
splatSimPathDE (splatSimDE), 31
splatSimSingleCellMeans
   (splatSimCellMeans), 31
splatSimTrueCounts, 33, 35
splatSimulate, 30, 34
splatSimulateGroups (splatSimulate), 34
splatSimulatePaths (splatSimulate), 34
splatSimulateSingle (splatSimulate), 34
splatter, 36
splatter-package (splatter), 36
SummarizedExperiment, 18

winsorize, 36