

# Package ‘splatter’

September 22, 2020

**Type** Package

**Title** Simple Simulation of Single-cell RNA Sequencing Data

**Version** 1.12.0

**Date** 2020-04-22

**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. Parameters can be estimated from real data and functions are provided for comparing real and simulated datasets.

**License** GPL-3 + file LICENSE

**LazyData** TRUE

**Depends** R (>= 4.0), SingleCellExperiment

**Imports** akima, BiocGenerics, BiocParallel, checkmate (>= 2.0.0), edgeR, fitdistrplus, ggplot2, locfit, matrixStats, methods, scales, scatter (>= 1.15.16), stats, SummarizedExperiment, utils, crayon, S4Vectors

**Suggests** BiocStyle, covr, cowplot, knitr, limSolve, lme4, progress, pscl, testthat, rmarkdown, scDD, scran, mfa, phenopath, BASiCS (>= 1.7.10), zinbwave, SparseDC, BiocManager, spelling, igraph, DropletUtils, BiocSingular

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

**URL** <https://github.com/Oshlack/splatter>

**BugReports** <https://github.com/Oshlack/splatter/issues>

**RoxygenNote** 7.1.0

**Encoding** UTF-8

**VignetteBuilder** knitr

**Language** en-GB

**git\_url** <https://git.bioconductor.org/packages/splatter>

**git\_branch** RELEASE\_3\_11

**git\_last\_commit** 259b2df

**git\_last\_commit\_date** 2020-04-27

**Date/Publication** 2020-09-21

**Author** Luke Zappia [aut, cre] (<<https://orcid.org/0000-0001-7744-8565>>),  
 Belinda Phipson [aut] (<<https://orcid.org/0000-0002-1711-7454>>),  
 Alicia Oshlack [aut] (<<https://orcid.org/0000-0001-9788-5690>>)

**Maintainer** Luke Zappia <[luke@lazappi.id.au](mailto:luke@lazappi.id.au)>

## R topics documented:

addFeatureStats . . . . .	4
addGeneLengths . . . . .	5
BASiCSEstimate . . . . .	6
BASiCSParams . . . . .	8
BASiCSSimulate . . . . .	8
bridge . . . . .	9
bringItemsForward . . . . .	10
compareSCEs . . . . .	10
diffSCEs . . . . .	11
expandParams . . . . .	13
getBetaStepProbs . . . . .	13
getCounts . . . . .	14
getLNormFactors . . . . .	14
getParam . . . . .	15
getParams . . . . .	15
getPathOrder . . . . .	16
kersplatEstBCV . . . . .	16
kersplatEstimate . . . . .	17
kersplatEstLib . . . . .	18
kersplatEstMean . . . . .	18
kersplatGenNetwork . . . . .	19
KersplatParams . . . . .	20
kersplatSample . . . . .	21
kersplatSelectRegs . . . . .	22
kersplatSetup . . . . .	23
kersplatSimAmbientCounts . . . . .	24
kersplatSimCellCounts . . . . .	24
kersplatSimCellMeans . . . . .	25
kersplatSimCounts . . . . .	26
kersplatSimGeneMeans . . . . .	26
kersplatSimLibSizes . . . . .	27
kersplatSimPaths . . . . .	28
kersplatSimulate . . . . .	29
listSims . . . . .	29
logistic . . . . .	30
lun2Estimate . . . . .	30
Lun2Params . . . . .	32
lun2Simulate . . . . .	33
lunEstimate . . . . .	34
LunParams . . . . .	35
lunSimulate . . . . .	35
makeCompPanel . . . . .	36
makeDiffPanel . . . . .	37
makeOverallPanel . . . . .	38

mfaEstimate	38
MFAParams	39
mfaSimulate	40
newParams	41
Params	42
phenoEstimate	42
PhenoParams	43
phenoSimulate	43
rbindMatched	44
sampleDensity	45
scDDEstimate	45
SCDDParams	47
scDDSimulate	48
selectFit	49
setParam	49
setParams	50
setParamsUnchecked	51
setParamUnchecked	52
showDFs	52
showPP	53
showValues	53
simpleEstimate	54
SimpleParams	55
simpleSimulate	55
sparseDCEstimate	56
SparseDCParams	57
sparseDCSimulate	58
splatEstBCV	59
splatEstDropout	59
splatEstimate	60
splatEstLib	61
splatEstMean	61
splatEstOutlier	62
SplatParams	62
splatSimBatchCellMeans	64
splatSimBatchEffects	64
splatSimBCVMeans	65
splatSimCellMeans	65
splatSimDE	66
splatSimDropout	66
splatSimGeneMeans	67
splatSimLibSizes	67
splatSimTrueCounts	68
splatSimulate	68
splatter	71
summariseDiff	71
summariseKS	72
summariseStats	72
tidyStatSumm	73
winsorize	73
zinbEstimate	74
ZINBParams	75

zinbSimulate . . . . . 76

**Index** . . . . . 78

addFeatureStats      *Add feature statistics*

## Description

Add additional feature statistics to a SingleCellExperiment object

## Usage

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

## Arguments

sce	SingleCellExperiment 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 SingleCellExperiment.
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 `rowData` 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 analysis packages.

## Value

SingleCellExperiment with additional feature statistics

---

addGeneLengths	<i>Add gene lengths</i>
----------------	-------------------------

---

### Description

Add gene lengths to an `SingleCellExperiment` object

### Usage

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

### Arguments

<code>sce</code>	SingleCellExperiment to add gene lengths to.
<code>method</code>	Method to use for creating lengths.
<code>loc</code>	Location parameter for the generate method.
<code>scale</code>	Scale parameter for the generate method.
<code>lengths</code>	Vector of lengths for the sample method.

### Details

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

### Value

SingleCellExperiment with added gene lengths

### Examples

```
# Default generate method  
sce <- simpleSimulate()  
sce <- addGeneLengths(sce)  
head(rowData(sce))  
# Sample method (human coding genes)  
## Not run:  
library(TxDb.Hsapiens.UCSC.hg19.knownGene)  
library(GenomicFeatures)  
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene  
tx.lens <- transcriptLengths(txdb, with.cds_len = TRUE)  
tx.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)
```

---

BASiCSEstimate

*Estimate BASiCS simulation parameters*

---

## Description

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

## Usage

```
BASiCSEstimate(
  counts,
  spike.info = NULL,
  batch = NULL,
  n = 20000,
  thin = 10,
  burn = 5000,
  regression = TRUE,
  params = newBASiCSParams(),
  verbose = TRUE,
  progress = TRUE,
  ...
)

## S3 method for class 'SingleCellExperiment'
BASiCSEstimate(
  counts,
  spike.info = NULL,
  batch = NULL,
  n = 20000,
  thin = 10,
  burn = 5000,
  regression = TRUE,
  params = newBASiCSParams(),
  verbose = TRUE,
  progress = TRUE,
  ...
)

## S3 method for class 'matrix'
BASiCSEstimate(
  counts,
  spike.info = NULL,
  batch = NULL,
  n = 20000,
  thin = 10,
  burn = 5000,
  regression = TRUE,
```

```

    params = newBASiCSParams(),
    verbose = TRUE,
    progress = TRUE,
    ...
  )

```

### Arguments

counts	either a counts matrix or a SingleCellExperiment object containing count data to estimate parameters from.
spike.info	data.frame describing spike-ins with two columns: "Name" giving the names of the spike-in features (must match rownames(counts)) and "Input" giving the number of input molecules.
batch	vector giving the batch that each cell belongs to.
n	total number of MCMC iterations. Must be $\geq \max(4, \text{thin})$ and a multiple of thin.
thin	thinning period for the MCMC sampler. Must be $\geq 2$ .
burn	burn-in period for the MCMC sampler. Must be in the range $1 \leq \text{burn} < n$ and a multiple of thin.
regression	logical. Whether to use regression to identify over-dispersion. See <a href="#">BASiCS_MCMC</a> for details.
params	BASiCSParams object to store estimated values in.
verbose	logical. Whether to print progress messages.
progress	logical. Whether to print additional BASiCS progress messages.
...	Optional parameters passed to <a href="#">BASiCS_MCMC</a> .

### Details

This function is just a wrapper around [BASiCS\\_MCMC](#) that takes the output and converts it to a BASiCSParams object. Either a set of spike-ins or batch information (or both) must be supplied. If only batch information is provided there must be at least two batches. See [BASiCS\\_MCMC](#) for details.

### Value

BASiCSParams object containing the estimated parameters.

### Examples

```

## Not run:
# Load example data
library(scater)
set.seed(1)
sce <- mockSCE()

spike.info <- data.frame(Name = rownames(sce)[1:10],
                        Input = rnorm(10, 500, 200),
                        stringsAsFactors = FALSE)
params <- BASiCSEstimate(sce[1:100, 1:30], spike.info)
params

## End(Not run)

```

---

 BASiCSParams

*The BASiCSParams class*


---

### Description

S4 class that holds parameters for the BASiCS simulation.

### Parameters

The BASiCS 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.

**Batch parameters** `nBatches` Number of batches to simulate.

`batchCells` Number of cells in each batch.

**Gene parameters** `gene.params` A data.frame containing gene parameters with two columns: Mean (mean expression for each biological gene) and Delta (cell-to-cell heterogeneity for each biological gene).

**Spike-in parameters** `nSpikes` The number of spike-ins to simulate.

`spike.means` Input molecules for each spike-in.

**Cell parameters** `cell.params` A data.frame containing gene parameters with two columns: Phi (mRNA content factor for each cell, scaled to sum to the number of cells in each batch) and S (capture efficient for each cell).

**Variability parameters** `theta` Technical variability parameter for each batch.

The parameters not shown in brackets can be estimated from real data using [BASiCSEstimate](#). For details of the BASiCS simulation see [BASiCSSimulate](#).

---

 BASiCSSimulate

*BASiCS simulation*


---

### Description

Simulate counts using the BASiCS method.

### Usage

```
BASiCSSimulate(params = newBASiCSParams(), verbose = TRUE, ...)
```

### Arguments

`params` BASiCSParams object containing simulation parameters.

`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 `BASiCS_Sim` that takes a `BASiCSParams`, runs the simulation then converts the output to a `SingleCellExperiment` object. See `BASiCS_Sim` for more details of how the simulation works.

## Value

`SingleCellExperiment` containing simulated counts

## References

Vallejos CA, Marioni JC, Richardson S. BASiCS: Bayesian Analysis of Single-Cell Sequencing data. *PLoS Computational Biology* (2015).

Paper: [10.1371/journal.pcbi.1004333](https://doi.org/10.1371/journal.pcbi.1004333)

Code: <https://github.com/catavallejos/BASiCS>

## Examples

```
if (requireNamespace("BASiCS", quietly = TRUE)) {  
  sim <- BASiCSSimulate()  
}
```

---

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)
```

## Arguments

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

## Value

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

---

bringItemsForward	<i>Bring items forward</i>
-------------------	----------------------------

---

**Description**

Move selected items to the start of a list.

**Usage**

```
bringItemsForward(ll, items)
```

**Arguments**

ll	list to adjust item order.
items	vector of items to bring to the front. Any not in the list will be ignored.

**Value**

list with selected items first

---

compareSCEs	<i>Compare SingleCellExperiment objects</i>
-------------	---

---

**Description**

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

**Usage**

```
compareSCEs(
  sces,
  point.size = 0.1,
  point.alpha = 0.1,
  fits = TRUE,
  colours = NULL
)
```

**Arguments**

sces	named list of SingleCellExperiment objects to combine and compare.
point.size	size of points in scatter plots.
point.alpha	opacity of points in scatter plots.
fits	whether to include fits in scatter plots.
colours	vector of colours to use for each dataset.

**Details**

The returned list has three items:

RowData Combined row data from the provided SingleCellExperiments.

ColData Combined column data from the provided SingleCellExperiments.

Plots Comparison 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-zeros relationship.

VarGeneCor Heatmap of correlation of the 100 most variable genes.

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**

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

---

diffSCEs

*Diff SingleCellExperiment objects*


---

**Description**

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

**Usage**

```
diffSCEs(
  sces,
  ref,
  point.size = 0.1,
  point.alpha = 0.1,
  fits = TRUE,
  colours = NULL
)
```

**Arguments**

<code>sces</code>	named list of <code>SingleCellExperiment</code> objects to combine and compare.
<code>ref</code>	string giving the name of the <code>SingleCellExperiment</code> to use as the reference
<code>point.size</code>	size of points in scatter plots.
<code>point.alpha</code>	opacity of points in scatter plots.
<code>fits</code>	whether to include fits in scatter plots.
<code>colours</code>	vector of colours to use for each dataset.

**Details**

This function aims to look at the differences between a reference `SingleCellExperiment` and one or more others. It requires each `SingleCellExperiment` 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 `SingleCellExperiment` used as the reference.

**RowData** Combined feature data from the provided `SingleCellExperiments`.

**ColData** Combined column data from the provided `SingleCellExperiments`.

**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.

**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**

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

---

expandParams	<i>Expand parameters</i>
--------------	--------------------------

---

**Description**

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

**Usage**

```
expandParams(object, ...)

## S4 method for signature 'BASiCParams'
expandParams(object)

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

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

**Arguments**

object	object to expand.
...	additional arguments.

**Value**

Expanded object.

---

getBetaStepProbs	<i>Get Beta step probabilities</i>
------------------	------------------------------------

---

**Description**

Use a Beta distribution for set probabilities along a path

**Usage**

```
getBetaStepProbs(steps, alpha, beta)
```

**Arguments**

steps	Number of steps
alpha	Alpha parameter
beta	Beta parameter

**Details**

The density is sampled from a Beta distribution between 0 and 1. Infinite densities at edges are adjusted and then the values are scaled to give probabilities.

**Value**

Vector of probabilities

---

getCounts	<i>Get counts</i>
-----------	-------------------

---

**Description**

Get counts matrix from a SingleCellExperiment object. If counts is missing a warning is issued and the first assay is returned.

**Usage**

```
getCounts(sce)
```

**Arguments**

sce                    SingleCellExperiment object

**Value**

Counts matrix

---

getLNormFactors	<i>Get log-normal factors</i>
-----------------	-------------------------------

---

**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	<i>Get a parameter</i>
----------	------------------------

---

**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	<i>Get parameters</i>
-----------	-----------------------

---

**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	<i>Get path order</i>
--------------	-----------------------

---

**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 originates from.

**Value**

Vector giving the order to process paths in.

---

kersplatEstBCV	<i>Estimate Kersplat BCV parameters</i>
----------------	---

---

**Description**

Estimate Biological Coefficient of Variation (BCV) parameters for the Kersplat simulation

**Usage**

```
kersplatEstBCV(counts, params, verbose)
```

**Arguments**

counts              counts matrix.  
 params              KersplatParams object to store estimated values in.  
 verbose              logical. Whether to print progress messages

**Details**

The [estimateDisp](#) function is used to estimate the common dispersion across the dataset. An exponential correction is applied based on fitting an exponential relationship between simulated and estimated values. If this results in a negative dispersion a simpler linear correction is applied instead.

**Value**

KersplatParams object with estimated BCV parameters



---

kersplatEstimate      *Estimate Kersplat simulation parameters*

---

## Description

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

## Usage

```
kersplatEstimate(counts, params = newKersplatParams(), verbose = TRUE)

## S3 method for class 'SingleCellExperiment'
kersplatEstimate(counts, params = newKersplatParams(), verbose = TRUE)

## S3 method for class 'matrix'
kersplatEstimate(counts, params = newKersplatParams(), verbose = TRUE)
```

## Arguments

counts	either a counts matrix or a SingleCellExperiment object containing count data to estimate parameters from.
params	KersplatParams object to store estimated values in.
verbose	logical. Whether to print progress messages.

## Value

KersplatParams object containing the estimated parameters.

## See Also

[kersplatEstMean](#), [kersplatEstBCV](#), [kersplatEstLib](#)

## Examples

```
if (requireNamespace("igraph", quietly = TRUE)) {
  # Load example data
  library(scater)
  set.seed(1)
  sce <- mockSCE()

  params <- kersplatEstimate(sce)
  params
}
```

---

kersplatEstLib	<i>Estimate Kersplat library size parameters</i>
----------------	--

---

**Description**

Estimate the library size parameters for the Kersplat simulation

**Usage**

```
kersplatEstLib(counts, params, verbose)
```

**Arguments**

counts	counts matrix.
params	KersplatParams object to store estimated values in.
verbose	logical. Whether to print progress messages

**Details**

Parameters for the log-normal distribution are estimated by fitting the library sizes using [fitdist](#). All the fitting methods are tried and the fit with the best Cramer-von Mises statistic is selected. The density of the library sizes is also estimated using [density](#).

**Value**

KersplatParams object with library size parameters

---

kersplatEstMean	<i>Estimate Kersplat means</i>
-----------------	--------------------------------

---

**Description**

Estimate mean parameters for the Kersplat simulation

**Usage**

```
kersplatEstMean(norm.counts, params, verbose)
```

**Arguments**

norm.counts	library size normalised counts matrix.
params	KersplatParams object to store estimated values in.
verbose	logical. Whether to print progress messages

## Details

Parameters for the gamma distribution are estimated by fitting the mean normalised counts using `fitdist`. All the fitting methods are tried and the fit with the best Cramer-von Mises statistic is selected. The density of the means is also estimated using `density`.

Expression outlier genes are detected using the Median Absolute Deviation (MAD) from median method. If the log<sub>2</sub> mean expression of a gene is greater than two MADs above the median log<sub>2</sub> 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 the `fitdist` MLE method.

## Value

KersplatParams object with estimated means

---

kersplatGenNetwork	<i>Generate Kersplat gene network</i>
--------------------	---------------------------------------

---

## Description

Generate a gene network for the Kersplat simulation

## Usage

```
kersplatGenNetwork(params, verbose)
```

## Arguments

params	KersplatParams object containing simulation parameters.
verbose	logical. Whether to print progress messages

## Details

Currently a very simple approach is used which needs to be improved. A network is generated using the `sample_forestfire` function and edge weights are sampled from a standard normal distribution.

## Value

KersplatParams object with gene network

## Description

S4 class that holds parameters for the Kersplat simulation.

## Parameters

The Kersplat 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 parameters** `mean.shape` Shape parameter for the mean gamma distribution.

`mean.rate` Rate parameter for the mean gamma distribution.

`mean.outProb` Probability that a gene is an expression outlier.

`mean.outFacLoc` Location (meanlog) parameter for the expression outlier factor log-normal distribution.

`mean.outFacScale` Scale (sdlog) parameter for the expression outlier factor log-normal distribution.

`mean.dens` [density](#) object describing the log gene mean density.

[`mean.method`] Method to use for simulating gene means. Either "fit" to sample from a gamma distribution (with expression outliers) or "density" to sample from the provided density object.

[`mean.values`] Vector of means for each gene.

**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.

**Network parameters** [`network.graph`] Graph containing the gene network.

[`network.nRegs`] Number of regulators in the network.

**Paths parameters** [`paths.programs`] Number of expression programs.

[`paths.design`] `data.frame` describing path structure. See [kersplatSimPaths](#) for details.

**Library size parameters** `lib.loc` Location (meanlog) parameter for the library size log-normal distribution, or mean parameter if a normal distribution is used.

`lib.scale` Scale (sdlog) parameter for the library size log-normal distribution, or sd parameter if a normal distribution is used.

`lib.dens` [density](#) object describing the library size density.

[`lib.method`] Method to use for simulating library sizes. Either "fit" to sample from a log-normal distribution or "density" to sample from the provided density object.

**Design parameters** [`cells.design`] `data.frame` describing cell structure. See [kersplatSimCellMeans](#) for details.

**Doublet parameters** [`doublet.prop`] Proportion of cells that are doublets.

**Ambient parameters** [`ambient.scale`] Scaling factor for the library size log-normal distribution when generating ambient library sizes.

[`ambient.nEmpty`] Number of empty cells to simulate.

The parameters not shown in brackets can be estimated from real data using [kersplatEstimate](#). For details of the Kersplat simulation see [kersplatSimulate](#).

---

kersplatSample	<i>Kersplat sample</i>
----------------	------------------------

---

## Description

Sample cells for the Kersplat simulation

## Usage

```
kersplatSample(params, verbose = TRUE)
```

## Arguments

params	KersplatParams object containing simulation parameters.
verbose	logical. Whether to print progress messages

## Details

The second stage is a two-step Kersplat simulation is to generate cells based on a complete [KersplatParams](#) object. intermediate parameters.

The sampling process involves the following steps:

1. Simulate library sizes for each cell
2. Simulate means for each cell
3. Simulate endogenous counts for each cell
4. Simulate ambient counts for each cell
5. Simulate final counts for each cell

The final output is a [SingleCellExperiment](#) object that contains the simulated counts but also the values for various intermediate steps. These are stored in the [colData](#) (for cell specific information), [rowData](#) (for gene specific information) or [assays](#) (for gene by cell matrices) slots. This additional information includes:

[colData](#) **Cell** Unique cell identifier.

**Type** Whether the cell is a Cell, Doublet or Empty.

**CellLibSize** The expected number of endogenous counts for that cell.

**AmbientLibSize** The expected number of ambient counts for that cell.

**Path** The path the cell belongs to.

**Step** How far along the path each cell is.

**Path1** For doublets the path of the first partner in the doublet (otherwise NA).

**Step1** For doublets the step of the first partner in the doublet (otherwise NA).

**Path2** For doublets the path of the second partner in the doublet (otherwise NA).

**Step2** For doublets the step of the second partner in the doublet (otherwise NA).

[rowData](#) **Gene** Unique gene identifier.

**BaseMean** The base expression level for that gene.

**AmbientMean** The ambient expression level for that gene.

[assays](#) **CellMeans** The mean expression of genes in each cell after any differential expression and adjusted for expected library size.

**CellCounts** Endogenous count matrix.

**AmbientCounts** Ambient count matrix.

**counts** Final count matrix.

Values that have been added by Splatter are named using UpperCamelCase in order to differentiate them from the values added by analysis packages which typically use underscore\_naming.

### Value

SingleCellExperiment object containing the simulated counts and intermediate values.

### See Also

[kersplatSimLibSizes](#), [kersplatSimCellMeans](#), [kersplatSimCellCounts](#), [kersplatSimAmbientCounts](#), [kersplatSimCounts](#)

### Examples

```
if (requireNamespace("igraph", quietly = TRUE)) {
  params <- kersplatSetup()
  sim <- kersplatSample(params)
}
```

---

kersplatSelectRegs      *Select Kersplat regulators*

---

### Description

Select regulator genes in the gene network for a Kersplat simulation

### Usage

```
kersplatSelectRegs(params, verbose)
```

### Arguments

params                    KersplatParams object containing simulation parameters.  
 verbose                  logical. Whether to print progress messages

### Details

Regulators are randomly selected, weighted according to the difference between their out degree and in degree. This is an arbitrary weighting and may be improved or replaced in the future.

### Value

KersplatParams object with gene regulators

---

kersplatSetup	<i>Kersplat setup</i>
---------------	-----------------------

---

### Description

Setup the parameters required for the Kersplat simulation

### Usage

```
kersplatSetup(params = newKersplatParams(), verbose = TRUE, ...)
```

### Arguments

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

### Details

The first stage of a two-step Kersplat simulation is to generate some of the intermediate parameters. The resulting parameters allow multiple simulated datasets to be generated from the same biological structure (using [kersplatSample](#)). As with all the other parameters these values can be manually overwritten if desired.

The setup involves the following steps:

1. Generate a gene network (if not already present)
2. Select regulator genes (if not already present)
3. Simulate gene means (if not already present)
4. Simulate cell paths

The resulting [KersplatParams](#) object will have the following parameters set (if they weren't already).

- `mean.values`
- `network.graph`
- `network.regsSet`
- `paths.means`

See [KersplatParams](#) for more details about these parameters and the functions for the individual steps for more details about the process.

### Value

A complete [KersplatParams](#) object

### See Also

[kersplatGenNetwork](#), [kersplatSelectRegs](#), [kersplatSimGeneMeans](#), [kersplatSimPaths](#), [KersplatParams](#)

**Examples**

```
if (requireNamespace("igraph", quietly = TRUE)) {
  params <- kersplatSetup()
}
```

---

kersplatSimAmbientCounts

*Simulate Kersplat ambient counts*

---

**Description**

Simulate Kersplat ambient counts

**Usage**

```
kersplatSimAmbientCounts(sim, params, verbose)
```

**Arguments**

sim	SingleCellExperiment containing simulation.
params	KersplatParams object with simulation parameters.
verbose	logical. Whether to print progress messages

**Details**

The overall expression profile to calculated by averaging the cell counts of the (non-empty) cells. This is then multiplied by the ambient library sizes to get a mean for each cell. Counts are then sampled from a Poisson distribution using these means.

**Value**

SingleCellExperiment with ambient counts

---

kersplatSimCellCounts *Simulate Kersplat cell counts*

---

**Description**

Simulate cell counts for the Kersplat simulation

**Usage**

```
kersplatSimCellCounts(sim, params, verbose)
```

**Arguments**

sim	SingleCellExperiment containing simulation.
params	KersplatParams object with simulation parameters.
verbose	logical. Whether to print progress messages



**Details**

Counts are sampled from a Poisson distribution with lambda equal to the cell means matrix.

**Value**

SingleCellExperiment with cell counts

---

kersplatSimCellMeans *Simulate Kersplat cell means*

---

**Description**

Simulate endogenous counts for each cell in a Kersplat simulation

**Usage**

```
kersplatSimCellMeans(sim, params, verbose)
```

**Arguments**

sim	SingleCellExperiment containing simulation.
params	KersplatParams object with simulation parameters.
verbose	logical. Whether to print progress messages

**Details**

Cells are first assigned to a path and a step along that path. This is controlled by the `cells.design` parameter which is a `data.frame` with the columns "Path", "Probability", "Alpha" and "Beta". The Path field is an ID for each path and the Probability field is the probability that a cell will come from that path (must sum to 1). The Alpha and Beta parameters control the density of cells along the path. After they are assigned to paths the step for each cell is sampled from a Beta distribution with parameters `shape1` equals Alpha and `shape2` equals beta. This approach is very flexible and allows almost any distribution of cells along a path. The distribution can be viewed using `hist(rbeta(10000, Alpha, Beta), breaks = 100)`. Some useful combinations of parameters are:

Alpha = 1, Beta = 1 Uniform distribution along the path  
 Alpha = 0, Beta = 1 All cells at the start of the path.  
 Alpha = 1, Beta = 0 All cells at the end of the path.  
 Alpha = 0, Beta = 0 Cells only at each end of the path.  
 Alpha = 1, Beta = 2 Linear skew towards the start of the path  
 Alpha = 0.5, Beta = 1 Curved skew towards the start of the path  
 Alpha = 2, Beta = 1 Linear skew towards the end of the path  
 Alpha = 1, Beta = 0.5 Curved skew towards the end of the path  
 Alpha = 0.5, Beta = 0.5 Curved skew towards both ends of the path  
 Alpha = 0.5, Beta = 0.5 Curved skew away from both ends of the path

Once cells are assigned to paths and steps the correct means are extracted from the `paths.means` parameter and adjusted based on each cell's library size. An adjustment for BCV is then applied. Doublets are also simulated at this stage by selecting two path/step combinations and averaging the means.

**Value**

SingleCellExperiment with cell means

---

kersplatSimCounts      *Simulate Kersplat final counts*

---

**Description**

Simulate the final counts matrix for a Kersplat simulation

**Usage**

```
kersplatSimCounts(sim, params, verbose)
```

**Arguments**

sim	SingleCellExperiment containing simulation.
params	KersplatParams object with simulation parameters.
verbose	logical. Whether to print progress messages

**Details**

The cell counts matrix and ambient counts matrix are added together. The result is then downsampled to the cell library size (for cells and doublets) or the ambient library size (for empty cells) using the [downsampleMatrix](#) function.

**Value**

SingleCellExperiment with counts matrix

**See Also**

[downsampleMatrix](#)

---

kersplatSimGeneMeans      *Simulate Kersplat gene means*

---

**Description**

Simulate Kersplat gene means

**Usage**

```
kersplatSimGeneMeans(params, verbose)
```

**Arguments**

params	KersplatParams object containing simulation parameters.
verbose	logical. Whether to print progress messages

**Details**

Gene means are simulated in one of two ways depending on the value of the `mean.method` parameter.

If `mean.method` is "fit" (default) then means are sampled from a Gamma distribution with shape equals `mean.shape` and rate equals `mean.rate`. Expression outliers are then added by replacing some values with the median multiplied by a factor from a log-normal distribution. This is the same process used for the Splat simulation.

If `mean.method` is "density" then means are sampled from the density object in the `mean.density` parameter using a rejection sampling method. This approach is more flexible but may violate some statistical assumptions.

**Value**

KersplatParams object with gene means

---

kersplatSimLibSizes     *Simulate Kersplat library sizes*

---

**Description**

Generate library sizes for cells in the Kersplat simulation

**Usage**

```
kersplatSimLibSizes(sim, params, verbose)
```

**Arguments**

<code>sim</code>	SingleCellExperiment containing simulation.
<code>params</code>	KersplatParams object with simulation parameters.
<code>verbose</code>	logical. Whether to print progress messages

**Details**

Library sizes are simulated in one of two ways depending on the value of the `lib.method` parameter.

If `lib.method` is "fit" (default) then means are sampled from a log-normal distribution with `meanlog` equals `lib.loc` and `sdlog` equals `lib.scale`.

If `lib.method` is "density" then library sizes are sampled from the density object in the `lib.density` parameter using a rejection sampling method. This approach is more flexible but may violate some statistical assumptions.

Ambient library sizes are also generated from a log-normal distribution based on the parameters for the cell library size and adjusted using the `ambient.scale` parameter.

**Value**

SingleCellExperiment with library sizes

---

kersplatSimPaths	<i>Simulate Kersplat paths</i>
------------------	--------------------------------

---

### Description

Simulate gene means for each step along each path of a Kersplat simulation

### Usage

```
kersplatSimPaths(params, verbose)
```

### Arguments

params	KersplatParams object containing simulation parameters.
verbose	logical. Whether to print progress messages

### Details

The method of simulating paths is inspired by the method used in the PROSSTT simulation. Changes in expression are controlled by `paths.nPrograms` regulatory programs. Each of the regulatory genes in the gene network has some association with each program. This is analogous to there being changes in the environment (the programs) which are sensed by receptors (regulatory genes) and cause changes in expression downstream. For each path a random walk is generated for each program and the changes passed on to the regulatory genes. At each step the changes propagate through the network according to the weights on edges between genes. This algorithm is fairly simple but should result in correlation relationships between genes. However it is likely to be improved and adjusted in the future.

The path structure itself is specified by the `paths.design` parameter. This is a `data.frame` with three columns: "Path", "From", and "Steps". The Path field is an ID for each path while the Steps field controls the length of each path. Increasing the number of steps will increase the difference in expression between the ends of the paths. The From field sets the originating point of each path. For example a From of  $\emptyset, \emptyset, \emptyset$  would indicate three paths from the origin while a From of  $\emptyset, 1, 1$  would give a branching structure with Path 1 beginning at the origin and Path 2 and Path 3 beginning at the end of Path 1.

### Value

KersplatParams object with path means

### References

Papadopoulos N, Parra RG, Söding J. PROSSTT: probabilistic simulation of single-cell RNA-seq data for complex differentiation processes. *Bioinformatics* (2019). <https://doi.org/10.1093/bioinformatics/btz078>.

---

kersplatSimulate	<i>Kersplat simulation</i>
------------------	----------------------------

---

**Description**

Simulate scRNA-seq count data using the Kersplat model

**Usage**

```
kersplatSimulate(params = newKersplatParams(), verbose = TRUE, ...)
```

**Arguments**

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

**Details**

This function is for simulating data in a single step. It consists of a call to [kersplatSetup](#) followed by a call to [kersplatSample](#). Please see the documentation for those functions for more details of the individual steps.

**Value**

SingleCellExperiment containing simulated counts and intermediate values

**See Also**

[kersplatSetup](#), [kersplatSample](#)

**Examples**

```
if (requireNamespace("igraph", quietly = TRUE)) {  
  sim <- kersplatSimulate  
}
```

---

listSims	<i>List simulations</i>
----------	-------------------------

---

**Description**

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

**Usage**

```
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**

```
listSims()
```

---

<code>logistic</code>	<i>Logistic function</i>
-----------------------	--------------------------

---

**Description**

Implementation of the logistic function

**Usage**

```
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

---

<code>lun2Estimate</code>	<i>Estimate Lun2 simulation parameters</i>
---------------------------	--

---

**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 'SingleCellExperiment'
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 a SingleCellExperiment 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 <a href="#">BiocParallelParam</a> instance giving the parallel back-end to be used. Default is <a href="#">SerialParam</a> which uses a single core.

**Details**

See [Lun2Params](#) for more details on the parameters.

**Value**

LunParams object containing the estimated parameters.

**Examples**

```
## Not run:
# Load example data
library(scater)
set.seed(1)
sce <- mockSCE()

plates <- as.numeric(factor(colData(sce)$Mutation_Status))
params <- lun2Estimate(sce, plates, min.size = 20)
params

## End(Not run)
```

Lun2Params

*The Lun2Params class***Description**

S4 class that holds parameters for the Lun2 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.

**Gene parameters** `gene.params` A data.frame containing gene parameters with two columns: Mean (mean expression for each gene) and Disp (dispersion for each gene).

`zi.params` A data.frame containing zero-inflated gene parameters with three columns: Mean (mean expression for each gene), Disp (dispersion for each, gene), and Prop (zero proportion for each gene).

[`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.

**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	<i>Lun2 simulation</i>
--------------	------------------------

---

### 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, ...)
```

### 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.

If the number of genes to simulate differs from the number of provided gene parameters or the number of cells to simulate differs from the number of library sizes the relevant parameters will be sampled with a warning. This allows any number of genes or cells to be simulated regardless of the number in the dataset used in the estimation step but has the downside that some genes or cells may be simulated multiple times.

### Value

SingleCellExperiment containing simulated counts.

### References

Lun ATL, Marioni JC. Overcoming confounding plate effects in differential expression analyses of single-cell RNA-seq data. *Biostatistics* (2017).

Paper: [dx.doi.org/10.1093/biostatistics/kxw055](https://doi.org/10.1093/biostatistics/kxw055)

Code: <https://github.com/MarioniLab/PlateEffects2016>

### Examples

```
sim <- lun2Simulate()
```

---

lunEstimate	<i>Estimate Lun simulation parameters</i>
-------------	---

---

### Description

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

### Usage

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

### Arguments

counts	either a counts matrix or an SingleCellExperiment 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

```
# Load example data  
library(scater)  
set.seed(1)  
sce <- mockSCE()  
  
params <- lunEstimate(sce)  
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](#).

lunSimulate

*Lun simulation***Description**

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 ( $2^{\wedge} \text{rnorm}(\text{nCells}, \text{mean} = 0, \text{sd} = 0.5)$ ) and differential expression can be simulated with fixed fold changes.

See [LunParams](#) for details of the parameters.

**Value**

SingleCellExperiment 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).

Paper: [dx.doi.org/10.1186/s13059-016-0947-7](https://doi.org/10.1186/s13059-016-0947-7)

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

**Examples**

```
sim <- lunSimulate()
```

---

makeCompPanel	<i>Make comparison panel</i>
---------------	------------------------------

---

**Description**

Combine the plots from compareSCEs into a single panel.

**Usage**

```
makeCompPanel(
  comp,
  title = "Comparison",
  labels = c("Means", "Variance", "Mean-variance relationship", "Library size",
            "Zeros per gene", "Zeros per cell", "Mean-zeros relationship")
)
```

**Arguments**

comp	list returned by <a href="#">compareSCEs</a> .
title	title for the panel.
labels	vector of labels for each of the seven plots.

**Value**

Combined panel plot

## Examples

```
## Not run:
sim1 <- splatSimulate(nGenes = 1000, batchCells = 20)
sim2 <- simpleSimulate(nGenes = 1000, nCells = 20)
comparison <- compareSCEs(list(Splat = sim1, Simple = sim2))
panel <- makeCompPanel(comparison)

## End(Not run)
```

---

makeDiffPanel	<i>Make difference panel</i>
---------------	------------------------------

---

## Description

Combine the plots from diffSCEs into a single panel.

## Usage

```
makeDiffPanel(
  diff,
  title = "Difference comparison",
  labels = c("Means", "Variance", "Library size", "Zeros per cell", "Zeros per gene",
    "Mean-variance relationship", "Mean-zeros relationship")
)
```

## Arguments

diff	list returned by <code>diffSCEs</code> .
title	title for the panel.
labels	vector of labels for each of the seven sections.

## Value

Combined panel plot

## Examples

```
## Not run:
sim1 <- splatSimulate(nGenes = 1000, batchCells = 20)
sim2 <- simpleSimulate(nGenes = 1000, nCells = 20)
difference <- diffSCEs(list(Splat = sim1, Simple = sim2), ref = "Simple")
panel <- makeDiffPanel(difference)

## End(Not run)
```

---

makeOverallPanel	<i>Make overall panel</i>
------------------	---------------------------

---

### Description

Combine the plots from compSCEs and diffSCEs into a single panel.

### Usage

```
makeOverallPanel(
  comp,
  diff,
  title = "Overall comparison",
  row.labels = c("Means", "Variance", "Mean-variance relationship", "Library size",
    "Zeros per cell", "Zeros per gene", "Mean-zeros relationship")
)
```

### Arguments

comp	list returned by <a href="#">compareSCEs</a> .
diff	list returned by <a href="#">diffSCEs</a> .
title	title for the panel.
row.labels	vector of labels for each of the seven rows.

### Value

Combined panel plot

### Examples

```
## Not run:
sim1 <- splatSimulate(nGenes = 1000, batchCells = 20)
sim2 <- simpleSimulate(nGenes = 1000, nCells = 20)
comparison <- compareSCEs(list(Splat = sim1, Simple = sim2))
difference <- diffSCEs(list(Splat = sim1, Simple = sim2), ref = "Simple")
panel <- makeOverallPanel(comparison, difference)

## End(Not run)
```

---

mfaEstimate	<i>Estimate mfa simulation parameters</i>
-------------	---

---

### Description

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

**Usage**

```

mfaEstimate(counts, params = newMFAParams())

## S3 method for class 'SingleCellExperiment'
mfaEstimate(counts, params = newMFAParams())

## S3 method for class 'matrix'
mfaEstimate(counts, params = newMFAParams())

```

**Arguments**

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

params            MFAParams object to store estimated values in.

**Details**

The nGenes and nCells parameters are taken from the size of the input data. The dropout lambda parameter is estimate using [empirical\\_lambda](#). See [MFAParams](#) for more details on the parameters.

**Value**

MFAParams object containing the estimated parameters.

**Examples**

```

# Load example data
if (requireNamespace("mfa", quietly = TRUE)) {
  library(mfa)
  synth <- create_synthetic(C = 20, G = 5, zero_negative = TRUE,
                           model_dropout = TRUE)

  params <- mfaEstimate(synth$X)
  params
}

```

---

MFAParams

*The MFAParams class*


---

**Description**

S4 class that holds parameters for the mfa simulation.

**Parameters**

The mfa 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.

[trans.prop] Proportion of genes that show transient expression. These genes are briefly up or down-regulated before returning to their initial state

[zero.neg] Logical. Whether to set negative expression values to zero. This will zero-inflate the data.

[dropout.present] Logical. Whether to simulate dropout.

dropout.lambda Lambda parameter for the exponential dropout function.

The parameters not shown in brackets can be estimated from real data using `mfaEstimate`. See `create_synthetic` for more details about the parameters. For details of the Splatter implementation of the mfa simulation see `mfaSimulate`.

---

mfaSimulate

*MFA simulation*

---

## Description

Simulate a bifurcating pseudotime path using the mfa method.

## Usage

```
mfaSimulate(params = newMFAParams(), verbose = TRUE, ...)
```

## Arguments

params	MFAParams object containing simulation parameters.
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 `create_synthetic` that takes a `MFAParams`, runs the simulation then converts the output from log-expression to counts and returns a `SingleCellExperiment` object. See `create_synthetic` and the mfa paper for more details about how the simulation works.

## Value

SingleCellExperiment containing simulated counts

## References

Campbell KR, Yau C. Probabilistic modeling of bifurcations in single-cell gene expression data using a Bayesian mixture of factor analyzers. Wellcome Open Research (2017).

Paper: [10.12688/wellcomeopenres.11087.1](https://doi.org/10.12688/wellcomeopenres.11087.1)

Code: <https://github.com/kieranrcampbell/mfa>

## Examples

```
if (requireNamespace("mfa", quietly = TRUE)) {
  sim <- mfaSimulate()
}
```



---

newParams	<i>New Params</i>
-----------	-------------------

---

## Description

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

## Usage

```
newBASiCParams(...)  
newKersplatParams(...)  
newLun2Params(...)  
newLunParams(...)  
newMFAParams(...)  
newPhenoParams(...)  
newSCDDParams(...)  
newSimpleParams(...)  
newSparseDCParams(...)  
newSplatParams(...)  
newZINBParams(...)
```

## Arguments

... additional parameters passed to [setParams](#).

## Value

New Params object.

## Examples

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

---

Params	<i>The Params virtual class</i>
--------	---------------------------------

---

### 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 not shown in brackets can be estimated from real data.

---

phenoEstimate	<i>Estimate PhenoPath simulation parameters</i>
---------------	---

---

### Description

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

### Usage

```
phenoEstimate(counts, params = newPhenoParams())
```

```
## S3 method for class 'SingleCellExperiment'
phenoEstimate(counts, params = newPhenoParams())
```

```
## S3 method for class 'matrix'
phenoEstimate(counts, params = newPhenoParams())
```

### Arguments

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

params PhenoParams object to store estimated values in.

### Details

The nGenes and nCells parameters are taken from the size of the input data. The total number of genes is evenly divided into the four types. See [PhenoParams](#) for more details on the parameters.

### Value

PhenoParams object containing the estimated parameters.

**Examples**

```

if (requireNamespace("phenopath", quietly = TRUE)) {
  # Load example data
  library(scater)
  set.seed(1)
  sce <- mockSCE()

  params <- phenoEstimate(sce)
  params
}

```

PhenoParams

*The PhenoParams class***Description**

S4 class that holds parameters for the PhenoPath simulation.

**Parameters**

The PhenoPath 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.

`[n.de]` Number of genes to simulate from the differential expression regime

`[n.pst]` Number of genes to simulate from the pseudotime regime

`[n.pst.beta]` Number of genes to simulate from the pseudotime + beta interactions regime

`[n.de.pst.beta]` Number of genes to simulate from the differential expression + pseudotime + interactions regime

The parameters not shown in brackets can be estimated from real data using [phenoEstimate](#). For details of the PhenoPath simulation see [phenoSimulate](#).

phenoSimulate

*PhenoPath simulation***Description**

Simulate counts from a pseudotime trajectory using the PhenoPath method.

**Usage**

```
phenoSimulate(params = newPhenoParams(), verbose = TRUE, ...)
```

**Arguments**

`params` PhenoParams object containing simulation parameters.

`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 `simulate_phenopath` that takes a `PhenoParams`, runs the simulation then converts the output from log-expression to counts and returns a `SingleCellExperiment` object. The original simulated log-expression values are returned in the `LogExprs` assay. See `simulate_phenopath` and the PhenoPath paper for more details about how the simulation works.

**Value**

`SingleCellExperiment` containing simulated counts

**References**

Campbell K, Yau C. Uncovering genomic trajectories with heterogeneous genetic and environmental backgrounds across single-cells and populations. *bioRxiv* (2017).

Paper: [10.1101/159913](https://doi.org/10.1101/159913)

Code: <https://github.com/kieranrcampbell/phenopath>

**Examples**

```
if (requireNamespace("phenopath", quietly = TRUE)) {
  sim <- phenoSimulate()
}
```

---

rbindMatched	<i>Bind rows (matched)</i>
--------------	----------------------------

---

**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.

---

sampleDensity	<i>Sample density</i>
---------------	-----------------------

---

**Description**

Sample from a density object using rejection sampling

**Usage**

```
sampleDensity(n, dens, lower = 0)
```

**Arguments**

n	Number of values to sample
dens	Density object to sample from
lower	Lower x-axis bound on sampled values

**Details**

Random points (x and y) are generated inside the range of the density object. If they value is less than the density for that x value (and x is greater than lower) then that x value is retained. Ten thousand points are generated at a time until enough valid values have been sampled.

**Value**

Vector of sampled values

---

scDDEstimate	<i>Estimate scDD simulation parameters</i>
--------------	--

---

**Description**

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

**Usage**

```
scDDEstimate(
  counts,
  params = newSCDDParams(),
  verbose = TRUE,
  BPPARAM = SerialParam(),
  ...
)

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

```

    BPPARAM = SerialParam(),
    conditions,
    ...
)

## S3 method for class 'SingleCellExperiment'
scDDEstimate(
  counts,
  params = newSCDDParams(),
  verbose = TRUE,
  BPPARAM = SerialParam(),
  condition = "condition",
  ...
)

## Default S3 method:
scDDEstimate(
  counts,
  params = newSCDDParams(),
  verbose = TRUE,
  BPPARAM = SerialParam(),
  condition,
  ...
)

```

### Arguments

counts	either a counts matrix or a SingleCellExperiment object containing count data to estimate parameters from.
params	SCDDParams object to store estimated values in.
verbose	logical. Whether to show progress messages.
BPPARAM	A <a href="#">BiocParallelParam</a> instance giving the parallel back-end to be used. Default is <a href="#">SerialParam</a> which uses a single core.
...	further arguments passed to or from other methods.
conditions	Vector giving the condition that each cell belongs to. Conditions can be 1 or 2.
condition	String giving the column that represents biological group of interest.

### Details

This function applies [preprocess](#) to the counts then uses [scDD](#) to estimate the numbers of each gene type to simulate. The output is then converted to a SCDDParams object. See [preprocess](#) and [scDD](#) for details.

### Value

SCDDParams object containing the estimated parameters.

### Examples

```

if (requireNamespace("scDD", quietly = TRUE)) {
  library(scater)
  set.seed(1)

```

```

sce <- mockSCE(ncells = 20, ngenes = 100)

colData(sce)$condition <- sample(1:2, ncol(sce), replace = TRUE)
params <- scDDEstimate(sce, condition = "condition")
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` [SingleCellExperiment](#) 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.

`nEP` Number of EP 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 about the parameters. For details of the Splatter implementation of the scDD simulation see [scDDSimulate](#).

---

`scDDSimulate`*scDD simulation*

---

## Description

Simulate counts using the scDD method.

## Usage

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

## Arguments

<code>params</code>	SCDDParams object containing simulation parameters.
<code>plots</code>	logical. whether to generate scDD fold change and validation plots.
<code>plot.file</code>	File path to save plots as PDF.
<code>verbose</code>	logical. Whether to print progress messages
<code>BPPARAM</code>	A <a href="#">BiocParallelParam</a> instance giving the parallel back-end to be used. Default is <a href="#">SerialParam</a> which uses a single core.
<code>...</code>	any additional parameter settings to override what is provided in <code>params</code> .

## Details

This function is just a wrapper around [simulateSet](#) that takes a [SCDDParams](#), runs the simulation then converts the output to a [SingleCellExperiment](#) object. See [simulateSet](#) for more details about how the simulation works.

## Value

SingleCellExperiment containing simulated counts

## References

Korthauer KD, Chu L-F, Newton MA, Li Y, Thomson J, Stewart R, et al. A statistical approach for identifying differential distributions in single-cell RNA-seq experiments. *Genome Biology* (2016).

Paper: [10.1186/s13059-016-1077-y](https://doi.org/10.1186/s13059-016-1077-y)

Code: <https://github.com/kdkorthauer/scDD>

## Examples

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



---

selectFit	<i>Select fit</i>
-----------	-------------------

---

**Description**

Try a variety of fitting methods and select the best one

**Usage**

```
selectFit(data, distr, weights = NULL, verbose = TRUE)
```

**Arguments**

data	The data to fit
distr	Name of the distribution to fit
weights	Optional vector of weights
verbose	logical. Whether to print progress messages

**Details**

The distribution is fitted to the data using each of the [fitdist](#) fitting methods. The fit with the smallest Cramer-von Mises statistic is selected.

**Value**

The selected fit object

---

setParam	<i>Set a parameter</i>
----------	------------------------

---

**Description**

Function for setting parameter values.

**Usage**

```
setParam(object, name, value)

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

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

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

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

## S4 method for signature 'ZINBParams'
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

*Set parameters*

---

### Description

Set multiple parameters in a Params object.

### Usage

```
setParams(object, update = NULL, ...)

## S4 method for signature 'KersplatParams'
setParams(object, update = NULL, ...)

## S4 method for signature 'Params'
setParams(object, update = NULL, ...)

## S4 method for signature 'SplatParams'
setParams(object, update = NULL, ...)
```

**Arguments**

object	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      *Set parameters UNCHECKED*

---

**Description**

Set multiple parameters in a Params object.

**Usage**

```
setParamsUnchecked(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. **THE FINAL OBJECT IS NOT CHECKED FOR VALIDITY!**

**Value**

Params object with updated values.

---

<code>setParamUnchecked</code>	<i>Set a parameter UNCHECKED</i>
--------------------------------	----------------------------------

---

**Description**

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

**Usage**

```
setParamUnchecked(object, name, value)

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

**Arguments**

<code>object</code>	object to set parameter in.
<code>name</code>	name of the parameter to set.
<code>value</code>	value to set the parameter to.

**Value**

Object with new parameter value.

---

<code>showDFs</code>	<i>Show data.frame</i>
----------------------	------------------------

---

**Description**

Function used for pretty printing data.frame parameters.

**Usage**

```
showDFs(dfs, not.default)
```

**Arguments**

<code>dfs</code>	list of data.frames to show.
<code>not.default</code>	logical vector giving which have changed from the default.

**Value**

Print data.frame parameters

---

showPP	<i>Show pretty print</i>
--------	--------------------------

---

**Description**

Function used for pretty printing params object.

**Usage**

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

**Arguments**

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

**Value**

Print params object to console

---

showValues	<i>Show values</i>
------------	--------------------

---

**Description**

Function used for pretty printing scalar or vector parameters.

**Usage**

```
showValues(values, not.default)
```

**Arguments**

values	list of values to show.
not.default	logical vector giving which have changed from the default.

**Value**

Print values

---

simpleEstimate	<i>Estimate simple simulation parameters</i>
----------------	--

---

### Description

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

### Usage

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

## S3 method for class 'SingleCellExperiment'
simpleEstimate(counts, params = newSimpleParams())

## S3 method for class 'matrix'
simpleEstimate(counts, params = newSimpleParams())
```

### Arguments

counts	either a counts matrix or a SingleCellExperiment 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.

### Value

SimpleParams object containing the estimated parameters.

### Examples

```
# Load example data
library(scater)
set.seed(1)
sce <- mockSCE()

params <- simpleEstimate(sce)
params
```

---

SimpleParams	<i>The SimpleParams class</i>
--------------	-------------------------------

---

**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	<i>Simple simulation</i>
----------------	--------------------------

---

**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 `shape = mean.shape` and `rate = mean.rate`. Counts are then simulated from a negative binomial distribution with `mu = means` and `size = 1 / counts.disp`. See [SimpleParams](#) for more details of the parameters.

**Value**

SingleCellExperiment containing simulated counts

**Examples**

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

---

sparseDCEstimate	<i>Estimate SparseDC simulation parameters</i>
------------------	--

---

**Description**

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

**Usage**

```
sparseDCEstimate(
  counts,
  conditions,
  nclusters,
  norm = TRUE,
  params = newSparseDCParams()
)

## S3 method for class 'SingleCellExperiment'
sparseDCEstimate(
  counts,
  conditions,
  nclusters,
  norm = TRUE,
  params = newSparseDCParams()
)

## S3 method for class 'matrix'
sparseDCEstimate(
  counts,
  conditions,
  nclusters,
  norm = TRUE,
  params = newSparseDCParams()
)
```

**Arguments**

counts	either a counts matrix or an SingleCellExperiment object containing count data to estimate parameters from.
conditions	numeric vector giving the condition each cell belongs to.
nclusters	number of cluster present in the dataset.
norm	logical, whether to library size normalise counts before estimation. Set this to FALSE if counts is already normalised.
params	PhenoParams object to store estimated values in.



**Details**

The `nGenes` and `nCells` parameters are taken from the size of the input data. The counts are preprocessed using `pre_proc_data` and then parameters are estimated using `sparsedc_cluster` using lambda values calculated using `lambda1_calculator` and `lambda2_calculator`.

See `SparseDCParams` for more details on the parameters.

**Value**

SparseParams object containing the estimated parameters.

**Examples**

```
if (requireNamespace("SparseDC", quietly = TRUE)) {
  # Load example data
  library(scater)
  set.seed(1)
  sce <- mockSCE(ncells = 20, ngenes = 100)

  conditions <- sample(1:2, ncol(sce), replace = TRUE)

  params <- sparseDCEstimate(sce, conditions, nclusters = 3)
  params
}
```

---

SparseDCParams

*The SparseDCParams class*

---

**Description**

S4 class that holds parameters for the SparseDC simulation.

**Parameters**

The SparseDC simulation uses the following parameters:

`nGenes` The number of genes to simulate in each condition.

`nCells` The number of cells to simulate.

[`seed`] Seed to use for generating random numbers.

`markers.n` Number of marker genes to simulate for each cluster.

`markers.shared` Number of marker genes for each cluster shared between conditions. Must be less than or equal to `markers.n`.

[`markers.same`] Logical. Whether each cluster should have the same set of marker genes.

`clusts.c1` Numeric vector of clusters present in condition 1. The number of times a cluster is repeated controls the proportion of cells from that cluster.

`clusts.c2` Numeric vector of clusters present in condition 2. The number of times a cluster is repeated controls the proportion of cells from that cluster.

[`mean.lower`] Lower bound for cluster gene means.

[`mean.upper`] Upper bound for cluster gene means.

The parameters not shown in brackets can be estimated from real data using `sparseDCEstimate`. For details of the SparseDC simulation see `sparseDCSimulate`.

---

sparseDCSimulate	<i>SparseDC simulation</i>
------------------	----------------------------

---

## Description

Simulate counts from cluster in two conditions using the SparseDC method.

## Usage

```
sparseDCSimulate(params = newSparseDCParams(), verbose = TRUE, ...)
```

## Arguments

params	SparseDCParams object containing simulation parameters.
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 [sim\\_data](#) that takes a [SparseDCParams](#), runs the simulation then converts the output from log-expression to counts and returns a [SingleCellExperiment](#) object. The original simulated log-expression values are returned in the LogExprs assay. See [sim\\_data](#) and the SparseDC paper for more details about how the simulation works.

## Value

SingleCellExperiment containing simulated counts

## References

Campbell K, Yau C. Uncovering genomic trajectories with heterogeneous genetic and environmental backgrounds across single-cells and populations. *bioRxiv* (2017).

Barron M, Zhang S, Li J. A sparse differential clustering algorithm for tracing cell type changes via single-cell RNA-sequencing data. *Nucleic Acids Research* (2017).

Paper: [10.1093/nar/gkx1113](https://doi.org/10.1093/nar/gkx1113)

## Examples

```
if (requireNamespace("SparseDC", quietly = TRUE)) {  
  sim <- sparseDCSimulate()  
}
```

---

splatEstBCV	<i>Estimate Splat Biological Coefficient of Variation parameters</i>
-------------	--

---

**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,  $disp = 0.1 + 0.25 * edgeR.disp$ .

**Value**

SplatParams object with estimated values.

---

splatEstDropout	<i>Estimate Splat dropout parameters</i>
-----------------	--

---

**Description**

Estimate the midpoint and shape parameters for the logistic function used when simulating dropout.

**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. Note this is done on the experiment level, more granular (eg. group or cell) level dropout is not estimated.

**Value**

SplatParams object with estimated values.

---

splatEstimate	<i>Estimate Splat simulation parameters</i>
---------------	---

---

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

**Arguments**

counts	either a counts matrix or a SingleCellExperiment 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**

```
# Load example data  
library(scater)  
set.seed(1)  
sce <- mockSCE()  
  
params <- splatEstimate(sce)  
params
```

---

splatEstLib	<i>Estimate Splat library size parameters</i>
-------------	---

---

**Description**

The Shapiro-Wilks test is used to determine if the library sizes are normally distributed. If so a normal distribution is fitted to the library sizes, if not (most cases) a log-normal distribution is fitted and the estimated parameters are added to the params object. See [fitdist](#) for details on the fitting.

**Usage**

```
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	<i>Estimate Splat mean parameters</i>
--------------	---------------------------------------

---

**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**

Parameters 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	<i>Estimate Splat expression outlier parameters</i>
-----------------	---

---

### 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 log<sub>2</sub> mean expression of a gene is greater than two MADs above the median log<sub>2</sub> 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	<i>The SplatParams class</i>
-------------	------------------------------

---

### Description

S4 class that holds parameters for the Splat simulation.

### Parameters

The Splat simulation requires 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.

**Batch parameters** [nBatches] The number of batches to simulate.

[batchCells] Vector giving the number of cells in each batch.

[batch.facLoc] Location (meanlog) parameter for the batch effect factor log-normal distribution. Can be a vector.

[batch.facScale] Scale (sdlog) parameter for the batch effect factor log-normal distribution. Can be a vector.

**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, or mean parameter if a normal distribution is used.

lib.scale Scale (sdlog) parameter for the library size log-normal distribution, or sd parameter if a normal distribution is used.

lib.norm Logical. Whether to use a normal distribution for library sizes instead of a log-normal.

**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.

**Group parameters** [nGroups] The number of groups or paths to simulate.

[group.prob] Probability that a cell comes from a group.

**Differential expression parameters** [de.prob] Probability that a gene is differentially expressed in a group. Can be a vector.

[de.downProb] 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.type The type of dropout to simulate. "none" indicates no dropout, "experiment" is global dropout using the same parameters for every cell, "batch" uses the same parameters for every cell in each batch, "group" uses the same parameters for every cell in each groups and "cell" uses a different set of parameters for each cell.

dropout.mid Midpoint parameter for the dropout logistic function.

dropout.shape Shape 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.nSteps] Vector giving the number of steps to simulate along each path. If a single value is given it will be applied to all paths. This parameter was previously called path.length.

[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 and 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 Splat simulation see [splatSimulate](#).

splatSimBatchCellMeans

*Simulate batch means*

### Description

Simulate a mean for each gene in each cell incorporating batch effect factors.

### Usage

```
splatSimBatchCellMeans(sim, params)
```

### Arguments

sim	SingleCellExperiment to add batch means to.
params	SplatParams object with simulation parameters.

### Value

SingleCellExperiment with simulated batch means.

splatSimBatchEffects *Simulate batch effects*

### Description

Simulate batch effects. Batch effect factors for each batch are produced using [getLNormFactors](#) and these are added along with updated means for each batch.

### Usage

```
splatSimBatchEffects(sim, params)
```

### Arguments

sim	SingleCellExperiment to add batch effects to.
params	SplatParams object with simulation parameters.

### Value

SingleCellExperiment with simulated batch effects.



---

splatSimBCVMeans	<i>Simulate BCV means</i>
------------------	---------------------------

---

**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	SingleCellExperiment to add BCV means to.
params	SplatParams object with simulation parameters.

**Value**

SingleCellExperiment with simulated BCV means.

---

splatSimCellMeans	<i>Simulate cell means</i>
-------------------	----------------------------

---

**Description**

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	SingleCellExperiment to add cell means to.
params	SplatParams object with simulation parameters.

**Value**

SingleCellExperiment with added cell means.

---

splatSimDE	<i>Simulate group differential expression</i>
------------	---

---

**Description**

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	SingleCellExperiment to add differential expression to.
params	splatParams object with simulation parameters.

**Value**

SingleCellExperiment with simulated differential expression.

---

splatSimDropout	<i>Simulate dropout</i>
-----------------	-------------------------

---

**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	SingleCellExperiment to add dropout to.
params	SplatParams object with simulation parameters.

**Value**

SingleCellExperiment with simulated dropout and observed counts.

---

splatSimGeneMeans	<i>Simulate gene means</i>
-------------------	----------------------------

---

**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	SingleCellExperiment to add gene means to.
params	SplatParams object with simulation parameters.

**Value**

SingleCellExperiment with simulated gene means.

---

splatSimLibSizes	<i>Simulate library sizes</i>
------------------	-------------------------------

---

**Description**

Simulate expected library sizes. Typically a log-normal distribution is used but there is also the option to use a normal distribution. In this case any negative values are set to half the minimum non-zero value.

**Usage**

```
splatSimLibSizes(sim, params)
```

**Arguments**

sim	SingleCellExperiment to add library size to.
params	SplatParams object with simulation parameters.

**Value**

SingleCellExperiment 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 it's own mean based on the group (or path position), expected library size and BCV.

### Usage

```
splatSimTrueCounts(sim, params)
```

### Arguments

sim                      SingleCellExperiment to add true counts to.  
 params                  SplatParams object with simulation parameters.

### Value

SingleCellExperiment with simulated true counts.

---

splatSimulate              *Splat simulation*

---

### Description

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

### Usage

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

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

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

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

**Arguments**

params	SplatParams object containing parameters for the simulation. See <a href="#">SplatParams</a> 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 dataset

The final output is a [SingleCellExperiment](#) object that contains the simulated counts but also the values for various intermediate steps. These are stored in the [colData](#) (for cell specific information), [rowData](#) (for gene specific information) or [assays](#) (for gene by cell matrices) slots. This additional information includes:

[colData](#) **Cell** Unique cell identifier.

**Group** The group or path the cell belongs to.

**ExpLibSize** The expected library size for that cell.

**Step (paths only)** how far along the path each cell is.

[rowData](#) **Gene** Unique gene identifier.

**BaseGeneMean** The base expression level for that gene.

**OutlierFactor** Expression outlier factor for that gene. Values of 1 indicate the gene is not an expression outlier.

**GeneMean** Expression level after applying outlier factors.

**BatchFac[Batch ]** The batch effects factor for each gene for a particular batch.

**DEFac[Group ]** The differential expression factor for each gene in a particular group. Values of 1 indicate the gene is not differentially expressed.

**SigmaFac[Path ]** Factor applied to genes that have non-linear changes in expression along a path.

[assays](#) **BatchCellMeans** The mean expression of genes in each cell after adding batch effects.

**BaseCellMeans** The mean expression of genes in each cell after any differential expression and adjusted for expected library size.

**BCV** The Biological Coefficient of Variation for each gene in each cell.

**CellMeans** The mean expression level of genes in each cell adjusted for BCV.

**TrueCounts** The simulated counts before dropout.

**Dropout** Logical matrix showing which values have been dropped in which cells.

Values that have been added by Splatter are named using UpperCamelCase in order to differentiate them from the values added by analysis packages which typically use underscore\_naming.

## Value

SingleCellExperiment object containing the simulated counts and intermediate values.

## References

Zappia L, Phipson B, Oshlack A. Splatter: simulation of single-cell RNA sequencing data. *Genome Biology* (2017).

Paper: [10.1186/s13059-017-1305-0](https://doi.org/10.1186/s13059-017-1305-0)

Code: <https://github.com/Oshlack/splatter>

## See Also

[splatSimLibSizes](#), [splatSimGeneMeans](#), [splatSimBatchEffects](#), [splatSimBatchCellMeans](#), [splatSimDE](#), [splatSimCellMeans](#), [splatSimBCVMeans](#), [splatSimTrueCounts](#), [splatSimDropout](#)

## Examples

```
# Simulation with default parameters
sim <- splatSimulate()
## Not run:
# 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)
```

---

`splatter`*splatter.*

---

### Description

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

### Details

As well as it's own simulation model **splatter** provides functions for the estimation of model parameters.

### See Also

Zappia L, Phipson B, Oshlack A. Splatter: Simulation Of Single-Cell RNA Sequencing Data. bioRxiv. 2017; doi:10.1101/133173

---

`summariseDiff`*Summarise diffSCEs*

---

### Description

Summarise the results of [diffSCEs](#). Calculates the Median Absolute Deviation (MAD), Mean Absolute Error (MAE), Root Mean Squared Error (RMSE) and Kolmogorov-Smirnov (KS) statistics for the various properties and ranks them.

### Usage

```
summariseDiff(diff)
```

### Arguments

`diff`                      Output from [diffSCEs](#)

### Value

data.frame with MADs, MAEs, RMSEs, scaled statistics and ranks

### Examples

```
sim1 <- splatSimulate(nGenes = 1000, batchCells = 20)
sim2 <- simpleSimulate(nGenes = 1000, nCells = 20)
difference <- diffSCEs(list(Splat = sim1, Simple = sim2), ref = "Simple")
summary <- summariseDiff(difference)
head(summary)
```

---

summariseKS	<i>Summarise KS</i>
-------------	---------------------

---

**Description**

Summarise columns of a data.frame compared to a reference using the KS test.

**Usage**

```
summariseKS(data, ref, split.col, stat.cols)
```

**Arguments**

data	The data.frame to summarise
ref	The reference data.frame
split.col	Name of the column used to split the dataset
stat.cols	Names of the columns to summarise. If this vector is named those names will be used in the output.

**Value**

data.frame with the summarised measure, scaled and ranked

---

summariseStats	<i>Summarise statistics</i>
----------------	-----------------------------

---

**Description**

Summarise columns of a data.frame using a single measure.

**Usage**

```
summariseStats(data, split.col, stat.cols, measure = c("MAD", "MAE", "RMSE"))
```

**Arguments**

data	The data.frame to summarise
split.col	Name of the column used to split the dataset
stat.cols	Names of the columns to summarise. If this vector is named those names will be used in the output.
measure	The measure to use for summarisation.

**Value**

data.frame with the summarised measure, scaled and ranked



---

tidyStatSumm	<i>Tidy summarised statistics</i>
--------------	-----------------------------------

---

**Description**

Convert a statistic summary to tidy format and add ranks and scaled values

**Usage**

```
tidyStatSumm(stat.summ, measure = c("MAD", "MAE", "RMSE"))
```

**Arguments**

stat.summ	The summary to convert
measure	The name of the summarisation measure

**Value**

tidy data.frame with the summarised measure, scaled and ranked

---

winsorize	<i>Winsorize vector</i>
-----------	-------------------------

---

**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

---

`zinbEstimate`*Estimate ZINB-WaVE simulation parameters*

---

**Description**

Estimate simulation parameters for the ZINB-WaVE simulation from a real dataset.

**Usage**

```
zinbEstimate(  
  counts,  
  design.samples = NULL,  
  design.genes = NULL,  
  common.disp = TRUE,  
  iter.init = 2,  
  iter.opt = 25,  
  stop.opt = 1e-04,  
  params = newZINBParams(),  
  verbose = TRUE,  
  BPPARAM = SerialParam(),  
  ...  
)  
  
## S3 method for class 'SingleCellExperiment'  
zinbEstimate(  
  counts,  
  design.samples = NULL,  
  design.genes = NULL,  
  common.disp = TRUE,  
  iter.init = 2,  
  iter.opt = 25,  
  stop.opt = 1e-04,  
  params = newZINBParams(),  
  verbose = TRUE,  
  BPPARAM = SerialParam(),  
  ...  
)  
  
## S3 method for class 'matrix'  
zinbEstimate(  
  counts,  
  design.samples = NULL,  
  design.genes = NULL,  
  common.disp = TRUE,  
  iter.init = 2,  
  iter.opt = 25,  
  stop.opt = 1e-04,  
  params = newZINBParams(),  
  verbose = TRUE,  
  BPPARAM = SerialParam(),  
  ...  
)
```

)

**Arguments**

counts	either a counts matrix or a SingleCellExperiment object containing count data to estimate parameters from.
design.samples	design matrix of sample-level covariates.
design.genes	design matrix of gene-level covariates.
common.disp	logical. Whether or not a single dispersion for all features is estimated.
iter.init	number of iterations to use for initialization.
iter.opt	number of iterations to use for optimization.
stop.opt	stopping criterion for optimization.
params	ZINBParams object to store estimated values in.
verbose	logical. Whether to print progress messages.
BPPARAM	A <a href="#">BiocParallelParam</a> instance giving the parallel back-end to be used. Default is <a href="#">SerialParam</a> which uses a single core.
...	additional arguments passes to <a href="#">zinbFit</a> .

**Details**

The function is a wrapper around [zinbFit](#) that takes the fitted model and inserts it into a [ZINBParams](#) object. See [ZINBParams](#) for more details on the parameters and [zinbFit](#) for details of the estimation procedure.

**Value**

ZINBParams object containing the estimated parameters.

**Examples**

```
if (requireNamespace("zinbwave", quietly = TRUE)) {
  library(scater)
  set.seed(1)
  sce <- mockSCE(ncells = 20, ngenes = 100)

  params <- zinbEstimate(sce)
  params
}
```

---

ZINBParams

*The ZINBParams class*


---

**Description**

S4 class that holds parameters for the ZINB-WaVE simulation.

## Parameters

The ZINB-WaVE 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.
- model Object describing a ZINB model.

The majority of the parameters for this simulation are stored in a [ZinbModel](#) object. Please refer to the documentation for this class and its constructor([zinbModel](#)) for details about all the parameters.

The parameters not shown in brackets can be estimated from real data using [zinbEstimate](#). For details of the ZINB-WaVE simulation see [zinbSimulate](#).

---

zinbSimulate	<i>ZINB-WaVE simulation</i>
--------------	-----------------------------

---

## Description

Simulate counts using the ZINB-WaVE method.

## Usage

```
zinbSimulate(params = newZINBParams(), verbose = TRUE, ...)
```

## Arguments

- params ZINBParams object containing simulation parameters.
- 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 [zinbSim](#) that takes a [ZINBParams](#), runs the simulation then converts the output to a [SingleCellExperiment](#) object. See [zinbSim](#) and the ZINB-WaVE paper for more details about how the simulation works.

## Value

SingleCellExperiment containing simulated counts

## References

Campbell K, Yau C. Uncovering genomic trajectories with heterogeneous genetic and environmental backgrounds across single-cells and populations. *bioRxiv* (2017).

Risso D, Perraudeau F, Gribkova S, Dudoit S, Vert J-P. ZINB-WaVE: A general and flexible method for signal extraction from single-cell RNA-seq data *bioRxiv* (2017).

Paper: [10.1101/125112](https://doi.org/10.1101/125112)

Code: <https://github.com/drisso/zinbwave>

**Examples**

```
if (requireNamespace("zinbwave", quietly = TRUE)) {  
  sim <- zinbSimulate()  
}
```

# Index

- addFeatureStats, 4
- addGeneLengths, 5
- assays, 21, 69
  
- BASiCS\_MCMC, 7
- BASiCS\_Sim, 9
- BASICSEstimate, 6, 8
- BASICSPParams, 8, 9
- BASICSPParams-class (BASICSPParams), 8
- BASICSSimulate, 8, 8
- BiocParallelParam, 31, 46, 48, 75
- bridge, 9
- bringItemsForward, 10
  
- colData, 21, 69
- compareSCEs, 10, 36, 38
- create\_synthetic, 40
  
- density, 18–20
- diffSCEs, 11, 37, 38, 71
- downsampleMatrix, 26
  
- empirical\_lambda, 39
- estimateDisp, 16, 59
- expandParams, 13
- expandParams, BASiCSParams-method (expandParams), 13
- expandParams, LunParams-method (expandParams), 13
- expandParams, SplatParams-method (expandParams), 13
  
- fitdist, 18, 19, 49, 54, 61, 62
  
- getBetaStepProbs, 13
- getCounts, 14
- getLNormFactors, 14, 64, 66
- getParam, 15
- getParam, Params-method (getParam), 15
- getParams, 15
- getPathOrder, 16
- ggplot, 11, 12
  
- kersplatEstBCV, 16, 17
- kersplatEstimate, 17, 20
- kersplatEstLib, 17, 18
- kersplatEstMean, 17, 18
- kersplatGenNetwork, 19, 23
- KersplatParams, 20, 21, 23
- KersplatParams-class (KersplatParams), 20
- kersplatSample, 21, 23, 29
- kersplatSelectRegs, 22, 23
- kersplatSetup, 23, 29
- kersplatSimAmbientCounts, 22, 24
- kersplatSimCellCounts, 22, 24
- kersplatSimCellMeans, 20, 22, 25
- kersplatSimCounts, 22, 26
- kersplatSimGeneMeans, 23, 26
- kersplatSimLibSizes, 22, 27
- kersplatSimPaths, 20, 23, 28
- kersplatSimulate, 20, 29
  
- lambda1\_calculator, 57
- lambda2\_calculator, 57
- listSims, 29
- logistic, 30
- lun2Estimate, 30, 32, 33
- Lun2Params, 31, 32
- Lun2Params-class (Lun2Params), 32
- lun2Simulate, 32, 33
- lunEstimate, 34, 35
- LunParams, 34, 35, 36
- LunParams-class (LunParams), 35
- lunSimulate, 35, 35
  
- makeCompPanel, 36
- makeDiffPanel, 37
- makeOverallPanel, 38
- mfaEstimate, 38, 40
- MFAPParams, 39, 39, 40
- MFAPParams-class (MFAPParams), 39
- mfaSimulate, 40, 40
  
- newBASiCSParams (newParams), 41
- newKersplatParams (newParams), 41
- newLun2Params (newParams), 41
- newLunParams (newParams), 41
- newMFAPParams (newParams), 41

- newParams, 41
- newPhenoParams (newParams), 41
- newSCDDParams (newParams), 41
- newSimpleParams (newParams), 41
- newSparseDCParams (newParams), 41
- newSplatParams (newParams), 41
- newZINBParams (newParams), 41
- nls, 59
- Params, 42
- Params-class (Params), 42
- phenoEstimate, 42, 43
- PhenoParams, 42, 43, 44
- PhenoParams-class (PhenoParams), 43
- phenoSimulate, 43, 43
- pre\_proc\_data, 57
- preprocess, 46
- rbindMatched, 44
- rowData, 4, 5, 21, 69
- sample\_forestfire, 19
- sampleDensity, 45
- scDD, 46
- scDDEstimate, 45, 47
- SCDDParams, 47, 48
- SCDDParams-class (SCDDParams), 47
- scDDSimulate, 47, 48
- selectFit, 49
- SerialParam, 31, 46, 48, 75
- setParam, 49, 51, 52
- setParam, BASiCParams-method (setParam), 49
- setParam, KersplatParams-method (setParam), 49
- setParam, Lun2Params-method (setParam), 49
- setParam, LunParams-method (setParam), 49
- setParam, Params-method (setParam), 49
- setParam, PhenoParams-method (setParam), 49
- setParam, SCDDParams-method (setParam), 49
- setParam, SplatParams-method (setParam), 49
- setParam, ZINBParams-method (setParam), 49
- setParams, 41, 50, 69
- setParams, KersplatParams-method (setParams), 50
- setParams, Params-method (setParams), 50
- setParams, SplatParams-method (setParams), 50
- setParamsUnchecked, 51
- setParamUnchecked, 52
- setParamUnchecked, Params-method (setParamUnchecked), 52
- showDFs, 52
- showPP, 53
- showValues, 53
- sim\_data, 58
- simpleEstimate, 54, 55
- SimpleParams, 54, 55, 55
- SimpleParams-class (SimpleParams), 55
- simpleSimulate, 55, 55
- simulate\_phenopath, 44
- simulateSet, 47, 48
- SingleCellExperiment, 5, 9, 21, 40, 44, 47, 48, 58, 69, 76
- sparsedc\_cluster, 57
- sparseDCEstimate, 56, 57
- SparseDCParams, 57, 57, 58
- SparseDCParams-class (SparseDCParams), 57
- sparseDCSimulate, 57, 58
- splatEstBCV, 59, 60
- splatEstDropout, 59, 60
- splatEstimate, 60, 64
- splatEstLib, 60, 61
- splatEstMean, 60, 61
- splatEstOutlier, 60, 62
- SplatParams, 62, 69
- SplatParams-class (SplatParams), 62
- splatSimBatchCellMeans, 64, 70
- splatSimBatchEffects, 64, 70
- splatSimBCVMeans, 65, 70
- splatSimCellMeans, 65, 70
- splatSimDE, 66, 70
- splatSimDropout, 66, 70
- splatSimGeneMeans, 67, 70
- splatSimGroupCellMeans (splatSimCellMeans), 65
- splatSimGroupDE (splatSimDE), 66
- splatSimLibSizes, 67, 70
- splatSimPathCellMeans (splatSimCellMeans), 65
- splatSimPathDE (splatSimDE), 66
- splatSimSingleCellMeans (splatSimCellMeans), 65
- splatSimTrueCounts, 68, 70
- splatSimulate, 64, 68
- splatSimulateGroups (splatSimulate), 68
- splatSimulatePaths (splatSimulate), 68
- splatSimulateSingle (splatSimulate), 68
- splatter, 71

summariseDiff, [71](#)  
summariseKS, [72](#)  
summariseStats, [72](#)  
  
tidyStatSumm, [73](#)  
  
winsorize, [73](#)  
  
zinbEstimate, [74](#), [76](#)  
zinbFit, [75](#)  
ZinbModel, [76](#)  
zinbModel, [76](#)  
ZINBParams, [75](#), [75](#), [76](#)  
ZINBParams-class (ZINBParams), [75](#)  
zinbSim, [76](#)  
zinbSimulate, [76](#), [76](#)