Introduction to Splatter

Luke Zappia

2018-12-06

Contents

Splatter logo

Welcome to Splatter! Splatter is an R package for the simple simulation of single-cell RNA sequencing data. This vignette gives an overview and introduction to Splatter's functionality.

1 Installation

Splatter can be installed from Bioconductor:

```
if (!requireNamespace("BiocManager", quietly=TRUE))
    install.packages("BiocManager")
BiocManager::install("splatter")
```

To install the most recent development version from Github use:

```
BiocManager::install("Oshlack/splatter", dependencies = TRUE,
         build_vignettes = TRUE)
```

2 Quickstart

Assuming you already have a matrix of count data similar to that you wish to simulate there are two simple steps to creating a simulated data set with Splatter. Here is an example using the example dataset in the `scater` package:

```
# Load package
library(splatter)
```

```
## Loading required package: SingleCellExperiment
```

```
## Loading required package: SummarizedExperiment
```

```
## Loading required package: GenomicRanges
```

```
## Loading required package: stats4
```

```
## Loading required package: BiocGenerics
```

```
## Loading required package: parallel
```

```
##
## Attaching package: 'BiocGenerics'
```

```
## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB
```

```
## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs
```

```
## The following objects are masked from 'package:base':
##
##     Filter, Find, Map, Position, Reduce, anyDuplicated, append,
##     as.data.frame, basename, cbind, colMeans, colSums, colnames,
##     dirname, do.call, duplicated, eval, evalq, get, grep, grepl,
##     intersect, is.unsorted, lapply, lengths, mapply, match, mget,
##     order, paste, pmax, pmax.int, pmin, pmin.int, rank, rbind,
##     rowMeans, rowSums, rownames, sapply, setdiff, sort, table,
##     tapply, union, unique, unsplit, which, which.max, which.min
```

```
## Loading required package: S4Vectors
```

```
##
## Attaching package: 'S4Vectors'

## The following object is masked from 'package:base':
##
##     expand.grid

## Loading required package: IRanges

## Loading required package: GenomeInfoDb

## Loading required package: Biobase

## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.

## Loading required package: DelayedArray

## Loading required package: matrixStats

##
## Attaching package: 'matrixStats'

## The following objects are masked from 'package:Biobase':
##
##     anyMissing, rowMedians

## Loading required package: BiocParallel

##
## Attaching package: 'DelayedArray'

## The following objects are masked from 'package:matrixStats':
##
##     colMaxs, colMins, colRanges, rowMaxs, rowMins, rowRanges

## The following objects are masked from 'package:base':
##
##     aperm, apply
```

```
# Load example data
library(scater)
```

```
## Loading required package: ggplot2

##
## Attaching package: 'scater'

## The following object is masked from 'package:S4Vectors':
##
##     rename

## The following object is masked from 'package:stats':
##
##     filter
```

```
data("sc_example_counts")
# Estimate parameters from example data
params <- splatEstimate(sc_example_counts)
# Simulate data using estimated parameters
sim <- splatSimulate(params)
```

```
## Getting parameters...
```

```
## Creating simulation object...
```

```
## Simulating library sizes...
```

```
## Simulating gene means...
```

```
## Simulating BCV...
```

```
## Simulating counts...
```

```
## Simulating dropout (if needed)...
```

```
## Done!
```

These steps will be explained in detail in the following sections but briefly the first step takes a dataset and estimates simulation parameters from it and the second step takes those parameters and simulates a new dataset.

3 The Splat simulation

Before we look at how we estimate parameters let's first look at how Splatter simulates data and what those parameters are. We use the term 'Splat' to refer to the Splatter's own simulation and differentiate it from the package itself. The core of the Splat model is a gamma-Poisson distribution used to generate a gene by cell matrix of counts. Mean expression levels for each gene are simulated from a gamma distribution (https://en.wikipedia.org/wiki/Gamma_distribution) and the Biological Coefficient of Variation is used to enforce a mean-variance trend before counts are simulated from a Poisson distribution (https://en.wikipedia.org/wiki/Poisson_distribution). Splat also allows you to simulate expression outlier genes (genes with mean expression outside the gamma distribution) and dropout (random knock out of counts based on mean expression). Each cell is given an expected library size (simulated from a log-normal distribution) that makes it easier to match to a given dataset.

Splat can also simulate differential expression between groups of different types of cells or differentiation paths between different cells types where expression changes in a continuous way. These are described further in the simulating counts section.

3.1 Parameters

The parameters required for the Splat simulation are briefly described here:

- **Global 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` - The number of cells in each batch.
  - `batch.facLoc` - Location (meanlog) parameter for the batch effects factor log-normal distribution.
  - `batch.facScale` - Scale (sdlog) parameter for the batch effects factor log-normal distribution.

- **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 for the normal distribution.
    - `lib.scale` - Scale (sdlog) parameter for the library size log-normal distribution, or sd for the normal distribution.
    - `lib.norm` - Whether to use a normal distribution instead of the usual log-normal distribution.
- **Expression outlier parameters**
    - `out.prob` - Probability that a gene is an expression outlier.
    - `out.facLoc` - Location (meanlog) parameter for the expression outlier factor log-normal distribution.
    - `out.facScale` - Scale (sdlog) parameter for the expression outlier factor log-normal distribution.
- **Group parameters**
    - `nGroups` - The number of groups or paths to simulate.
    - `group.prob` - The probabilities that cells come from particular groups.
- **Differential expression parameters**
    - `de.prob` - Probability that a gene is differentially expressed in each group or path.
    - `de.loProb` - Probability that a differentially expressed gene is down-regulated.
    - `de.facLoc` - Location (meanlog) parameter for the differential expression factor log-normal distribution.
    - `de.facScale` - Scale (sdlog) parameter for the differential expression factor log-normal distribution.
- **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` - Type of dropout to simulate.
    - `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.
    - `path.length` - Vector giving the number of steps to simulate along each path.
    - `path.skew` - Vector giving the skew of each path.
    - `path.nonlinearProb` - Probability that a gene changes expression in a non-linear way along the differentiation path.
    - `path.sigmaFac` - Sigma factor for non-linear gene paths.

While this may look like a lot of parameters Splatter attempts to make it easy for the user, both by providing sensible defaults and making it easy to estimate many of the parameters from real data. For more details on the parameters see `?SplatParams`.

4 The `SplatParams` object

All the parameters for the Splat simulation are stored in a `SplatParams` object. Let's create a new one and see what it looks like.

```
params <- newSplatParams()
params
```

```
## A Params object of class SplatParams
## Parameters can be (estimable) or [not estimable], 'Default' or  'NOT DEFA
ULT'
##
## Global:
## (Genes)  (Cells)   [Seed]
##   10000      100   616494
##
## 28 additional parameters
##
## Batches:
##     [Batches]  [Batch Cells]     [Location]        [Scale]
##             1            100            0.1            0.1
##
## Mean:
##  (Rate)  (Shape)
##     0.3      0.6
##
## Library size:
## (Location)     (Scale)      (Norm)
##         11         0.2       FALSE
##
## Exprs outliers:
## (Probability)     (Location)        (Scale)
##          0.05              4            0.5
##
## Groups:
##      [Groups]  [Group Probs]
##             1              1
##
## Diff expr:
## [Probability]    [Down Prob]     [Location]        [Scale]
##           0.1            0.5            0.1            0.4
##
## BCV:
## (Common Disp)          (DoF)
##           0.1             60
##
## Dropout:
##     [Type]  (Midpoint)     (Shape)
##       none           0          -1
##
## Paths:
##          [From]         [Length]           [Skew]    [Non-linear]
##               0              100              0.5             0.1
## [Sigma Factor]
##            0.8
```

As well as telling us what type of object we have ("A `Params` object of class `SplatParams` ") and showing us the values of the parameter this output gives us some extra information. We can see which parameters can be estimated by the `splatEstimate` function (those in parentheses), which can't be estimated (those in brackets) and which have been changed from their default values (those in ALL CAPS).

4.1 Getting and setting

If we want to look at a particular parameter, for example the number of genes to simulate, we can extract it using the `getParam` function:

```
getParam(params, "nGenes")
```

```
## [1] 10000
```

Alternatively, to give a parameter a new value we can use the `setParam` function:

```
params <- setParam(params, "nGenes", 5000)
getParam(params, "nGenes")
```

```
## [1] 5000
```

If we want to extract multiple parameters (as a list) or set multiple parameters we can use the `getParams` or `setParams` functions:

```
# Set multiple parameters at once (using a list)
params <- setParams(params, update = list(nGenes = 8000, mean.rate = 0.5))
# Extract multiple parameters as a list
getParams(params, c("nGenes", "mean.rate", "mean.shape"))
```

```
## $nGenes
## [1] 8000
##
## $mean.rate
## [1] 0.5
##
## $mean.shape
## [1] 0.6
```

```
# Set multiple parameters at once (using additional arguments)
params <- setParams(params, mean.shape = 0.5, de.prob = 0.2)
params
```

```
## A Params object of class SplatParams
## Parameters can be (estimable) or [not estimable], 'Default' or  'NOT DEFA
ULT'
##
## Global:
## (GENES)  (Cells)   [Seed]
##    8000      100   616494
##
## 28 additional parameters
##
## Batches:
##     [Batches]  [Batch Cells]     [Location]        [Scale]
##             1            100            0.1            0.1
##
## Mean:
##  (RATE)  (SHAPE)
##     0.5      0.5
##
## Library size:
## (Location)     (Scale)      (Norm)
##         11         0.2       FALSE
##
## Exprs outliers:
## (Probability)     (Location)        (Scale)
##          0.05              4            0.5
##
## Groups:
##      [Groups]  [Group Probs]
##             1              1
##
## Diff expr:
## [PROBABILITY]    [Down Prob]     [Location]        [Scale]
##           0.2            0.5            0.1            0.4
##
## BCV:
## (Common Disp)          (DoF)
##           0.1             60
##
## Dropout:
##     [Type]  (Midpoint)     (Shape)
##       none           0          -1
##
## Paths:
##         [From]         [Length]           [Skew]    [Non-linear]
##              0              100              0.5             0.1
## [Sigma Factor]
##            0.8
```

The parameters with have changed are now shown in ALL CAPS to indicate that they been changed form the default.

We can also set parameters directly when we call `newSplatParams` :

```
params <- newSplatParams(lib.loc = 12, lib.scale = 0.6)
getParams(params, c("lib.loc", "lib.scale"))
```

```
## $lib.loc
## [1] 12
##
## $lib.scale
## [1] 0.6
```

5 Estimating parameters

Splat allows you to estimate many of it's parameters from a data set containing counts using the `splatEstimate` function.

```
# Check that sc_example counts is an integer matrix
class(sc_example_counts)
```

```
## [1] "matrix"
```

```
typeof(sc_example_counts)
```

```
## [1] "integer"
```

```
# Check the dimensions, each row is a gene, each column is a cell
dim(sc_example_counts)
```

```
## [1] 2000   40
```

```
# Show the first few entries
sc_example_counts[1:5, 1:5]
```

```
##           Cell_001 Cell_002 Cell_003 Cell_004 Cell_005
## Gene_0001        0      123        2        0        0
## Gene_0002      575       65        3     1561     2311
## Gene_0003        0        0        0        0     1213
## Gene_0004        0        1        0        0        0
## Gene_0005        0        0       11        0        0
```

```
params <- splatEstimate(sc_example_counts)
```

Here we estimated parameters from a counts matrix but `splatEstimate` can also take a `SingleCellExperiment` object. The estimation process has the following steps:

1. Mean parameters are estimated by fitting a gamma distribution to the mean expression levels.
2. Library size parameters are estimated by fitting a log-normal distribution to the library sizes.
3. Expression outlier parameters are estimated by determining the number of outliers and fitting a log-normal distribution to their difference from the median.
4. BCV parameters are estimated using the `estimateDisp` function from the `edgeR` package.
5. Dropout parameters are estimated by checking if dropout is present and fitting a logistic function to the relationship between mean expression and proportion of zeros.

For more details of the estimation procedures see `?splatEstimate`.

6 Simulating counts

Once we have a set of parameters we are happy with we can use `splatSimulate` to simulate counts. If we want to make small adjustments to the parameters we can provide them as additional arguments, alternatively if we don't supply any parameters the defaults will be used:

```
sim <- splatSimulate(params, nGenes = 1000)
```

```
## Getting parameters...
```

```
## Creating simulation object...
```

```
## Simulating library sizes...
## Simulating gene means...
```

```
## Simulating BCV...
```

```
## Simulating counts...
```

```
## Simulating dropout (if needed)...
```

```
## Done!
```

```
sim
```

```
## class: SingleCellExperiment
## dim: 1000 40
## metadata(1): Params
## assays(6): BatchCellMeans BaseCellMeans ... TrueCounts counts
## rownames(1000): Gene1 Gene2 ... Gene999 Gene1000
## rowData names(4): Gene BaseGeneMean OutlierFactor GeneMean
## colnames(40): Cell1 Cell2 ... Cell39 Cell40
## colData names(3): Cell Batch ExpLibSize
## reducedDimNames(0):
## spikeNames(0):
```

Looking at the output of `splatSimulate` we can see that `sim` is `SingleCellExperiment` object with 1000 features (genes) and 40 samples (cells). The main part of this object is a features by samples matrix containing the simulated counts (accessed using `counts`), although it can also hold other expression measures such as FPKM or TPM. Additionaly a `SingleCellExperiment` contains phenotype information about each cell (accessed using `colData`) and feature information about each gene (accessed using `rowData`). Splatter uses these slots, as well as `assays`, to store information about the intermediate values of the simulation.

```
# Access the counts
counts(sim)[1:5, 1:5]
```

```
##       Cell1 Cell2 Cell3 Cell4 Cell5
## Gene1     6    24    52     0     0
## Gene2   152   357     3    57    37
## Gene3     0     0     0     0     0
## Gene4    65     0     0     0     0
## Gene5     0     0   328     0     0
```

```
# Information about genes
head(rowData(sim))
```

```
## DataFrame with 6 rows and 4 columns
##           Gene     BaseGeneMean OutlierFactor         GeneMean
##       <factor>        <numeric>     <numeric>        <numeric>
## Gene1    Gene1 83.3075155921279             1 83.3075155921279
## Gene2    Gene2 192.989877990138             1 192.989877990138
## Gene3    Gene3 5.34160546349346             1 5.34160546349346
## Gene4    Gene4 13.0996473659882             1 13.0996473659882
## Gene5    Gene5 47.5243352477243             1 47.5243352477243
## Gene6    Gene6 329.192230943705             1 329.192230943705
```

```
# Information about cells
head(colData(sim))
```

```
## DataFrame with 6 rows and 3 columns
##           Cell       Batch       ExpLibSize
##       <factor> <character>        <numeric>
## Cell1    Cell1      Batch1 147537.815840946
## Cell2    Cell2      Batch1 248570.943306099
## Cell3    Cell3      Batch1 570754.014539991
## Cell4    Cell4      Batch1 297469.621605139
## Cell5    Cell5      Batch1 465882.586413918
## Cell6    Cell6      Batch1 76235.7229277478
```

```
# Gene by cell matrices
names(assays(sim))
```

```
## [1] "BatchCellMeans" "BaseCellMeans"  "BCV"            "CellMeans"
## [5] "TrueCounts"     "counts"
```

```
# Example of cell means matrix
assays(sim)$CellMeans[1:5, 1:5]
```

```
##              Cell1        Cell2        Cell3        Cell4        Cell5
## Gene1 6.285409e+00 2.573702e+01  60.24793571 2.872454e-03 6.318224e-27
## Gene2 1.496247e+02 3.618770e+02   2.13897654 5.679536e+01 3.314663e+01
## Gene3 3.796172e-09 2.176432e-15   0.00117308 2.285623e-05 8.950682e-02
## Gene4 6.320246e+01 3.279620e-08   0.33118361 1.632851e-05 1.927820e-01
## Gene5 7.295955e-48 5.266185e-12 312.38438858 2.073377e-19 3.523634e-13
```

An additional (big) advantage of outputting a `SingleCellExperiment` is that we get immediate access to other analysis packages, such as the plotting functions in `scater`. For example we can make a PCA plot:

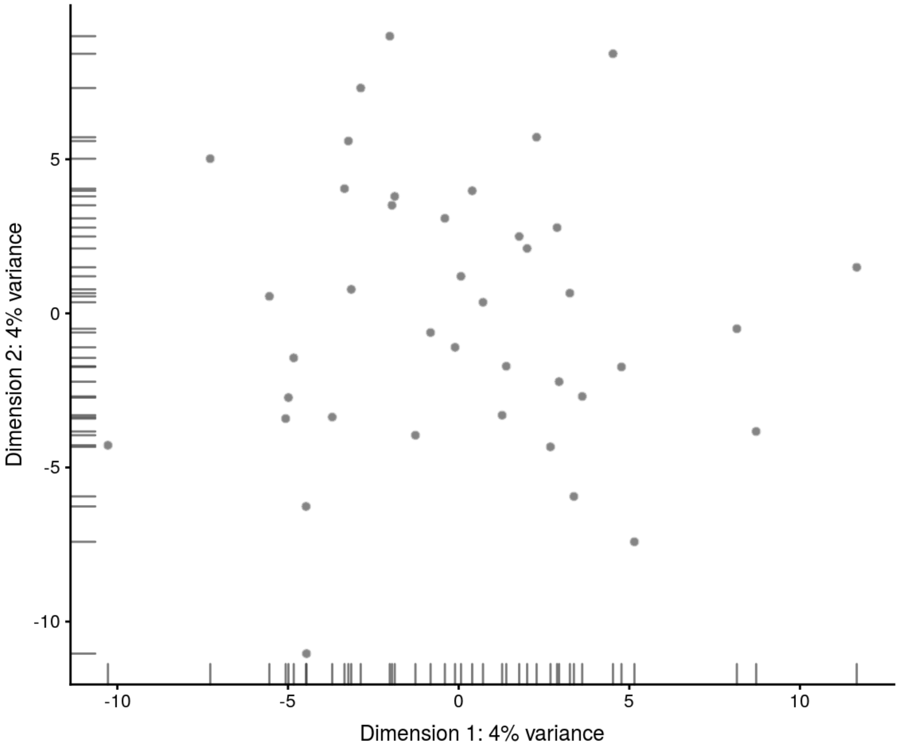
```
# Use scater to calculate logcounts
sim <- normalise(sim)
```

```
## Warning: 'normalise' is deprecated.
## Use 'normalize' instead.
## See help("Deprecated")
```

```
## Warning in .local(object, ...): using library sizes as size factors
```

```
# Plot PCA
plotPCA(sim)
```

```
## Warning: 'add_ticks' is deprecated.
## Use '+ geom_rug(...)' instead.
```



(**NOTE:** Your values and plots may look different as the simulation is random and produces different results each time it is run.)

For more details about the `SingleCellExperiment` object refer to the [vignette] SCE-vignette (https://bioconductor.org/packages/devel/bioc/vignettes/SingleCellExperiment/inst/doc/intro.html). For information about what you can do with `scater` refer to the `scater` documentation and vignette (https://bioconductor.org/packages/release/bioc/vignettes/scater/inst/doc/vignette.html).

The `splatSimulate` function outputs the following additional information about the simulation:

- **Cell information (`pData`)**
  - `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.
- **Gene information (`fData`)**
  - `Gene` - Unique gene identifier.
  - `BaseGeneMean` - The base expression level for that gene.
  - `OutlierFactor` - Expression outlier factor for that gene (1 is not an outlier).
  - `GeneMean` - Expression level after applying outlier factors.
  - `DEFac[Group]` - The differential expression factor for each gene in a particular group (1 is not differentially expressed).
  - `GeneMean[Group]` - Expression level of a gene in a particular group after applying differential expression factors.
- **Gene by cell information (`assayData`)**
  - `BaseCellMeans` - The expression of genes in each cell adjusted for expected library size.
  - `BCV` - The Biological Coefficient of Variation for each gene in each cell.
  - `CellMeans` - The expression level of genes in each cell adjusted for BCV.
  - `TrueCounts` - The simulated counts before dropout.
  - `Dropout` - Logical matrix showing which counts have been dropped in which cells.

Values that have been added by Splatter are named using `UpperCamelCase` to separate them from the `underscore_naming` used by `scater` and other packages. For more information on the simulation see `?splatSimulate` .

6.1 Simulating groups

So far we have only simulated a single population of cells but often we are interested in investigating a mixed population of cells and looking to see what cell types are present or what differences there are between them. Splatter is able to simulate these situations by changing the `method` argument Here we are going to simulate two groups, by specifying the `group.prob` parameter and setting the `method` parameter to `"groups"` :

(**NOTE:** We have also set the `verbose` argument to `FALSE` to stop Splatter printing progress messages.)

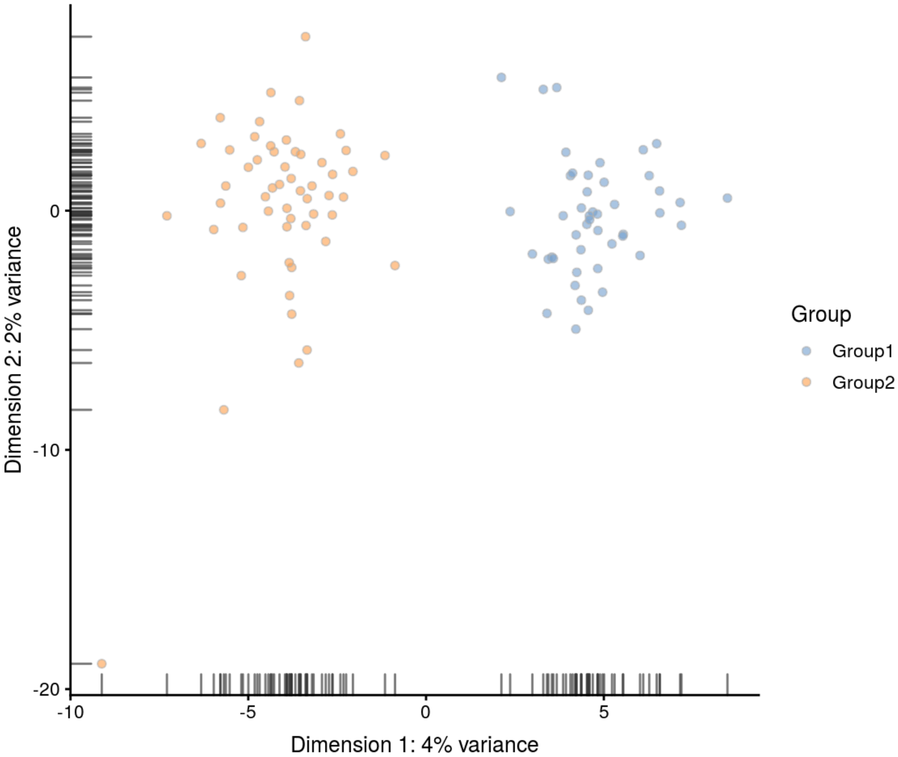
```
sim.groups <- splatSimulate(group.prob = c(0.5, 0.5), method = "groups",
                            verbose = FALSE)
sim.groups <- normalise(sim.groups)
```

```
## Warning: 'normalise' is deprecated.
## Use 'normalize' instead.
## See help("Deprecated")
```

```
## Warning in .local(object, ...): using library sizes as size factors
```

```
plotPCA(sim.groups, colour_by = "Group")
```

```
## Warning: 'add_ticks' is deprecated.
## Use '+ geom_rug(...)' instead.
```



As we have set both the group probabilites to 0.5 we should get approximately equal numbers of cells in each group (around 50 in this case). If we wanted uneven groups we could set `group.prob` to any set of probabilites that sum to 1.

6.2 Simulating paths

The other situation that is often of interest is a differentiation process where one cell type is changing into another. Splatter approximates this process by simulating a series of steps between two groups and randomly assigning each cell to a step. We can create this kind of simulation using the `"paths"` method.

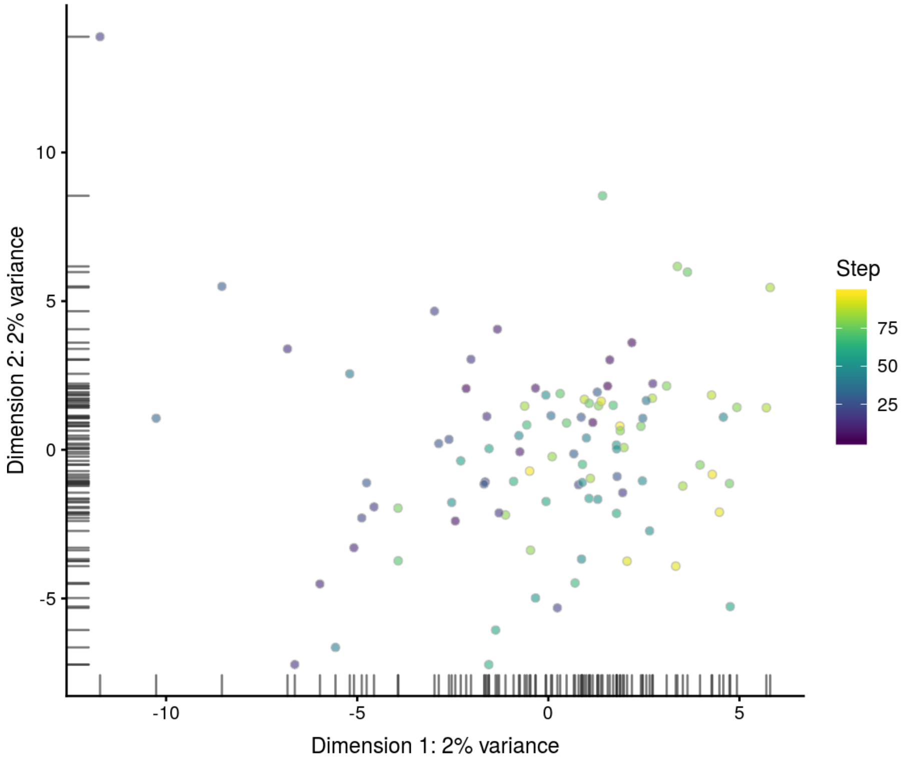
```
sim.paths <- splatSimulate(method = "paths", verbose = FALSE)
sim.paths <- normalise(sim.paths)
```

```
## Warning: 'normalise' is deprecated.
## Use 'normalize' instead.
## See help("Deprecated")
```

```
## Warning in .local(object, ...): using library sizes as size factors
```

```
plotPCA(sim.paths, colour_by = "Step")
```

```
## Warning: 'add_ticks' is deprecated.
## Use '+ geom_rug(...)' instead.
```



Here the colours represent the “step” of each cell or how far along the differentiation path it is. We can see that the cells with dark colours are more similar to the originating cell type and the light coloured cells are closer to the final, differentiated, cell type. By setting additional parameters it is possible to simulate more complex process (for example multiple mature cell types from a single progenitor).

6.3 Batch effects

Another factor that is important in the analysis of any sequencing experiment are batch effects, technical variation that is common to a set of samples processed at the same time. We apply batch effects by telling Splatter how many cells are in each batch:

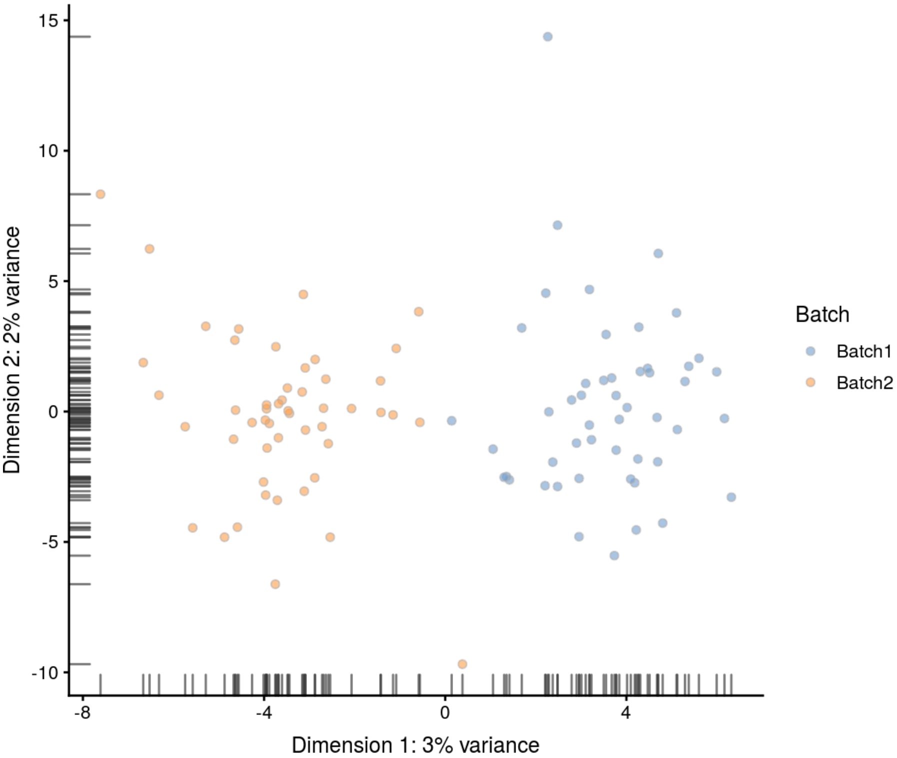
```
sim.batches <- splatSimulate(batchCells = c(50, 50), verbose = FALSE)
sim.batches <- normalise(sim.batches)
```

```
## Warning: 'normalise' is deprecated.
## Use 'normalize' instead.
## See help("Deprecated")
```

```
## Warning in .local(object, ...): using library sizes as size factors
```

```
plotPCA(sim.batches, colour_by = "Batch")
```

```
## Warning: 'add_ticks' is deprecated.
## Use '+ geom_rug(...)' instead.
```



This looks at lot like when we simulated groups and that is because the process is very similar. The difference is that batch effects are applied to all genes, not just those that are differentially expressed, and the effects are usually smaller. By combining groups and batches we can simulate both unwanted variation that we aren't interested in (batch) and the wanted variation we are looking for (group):

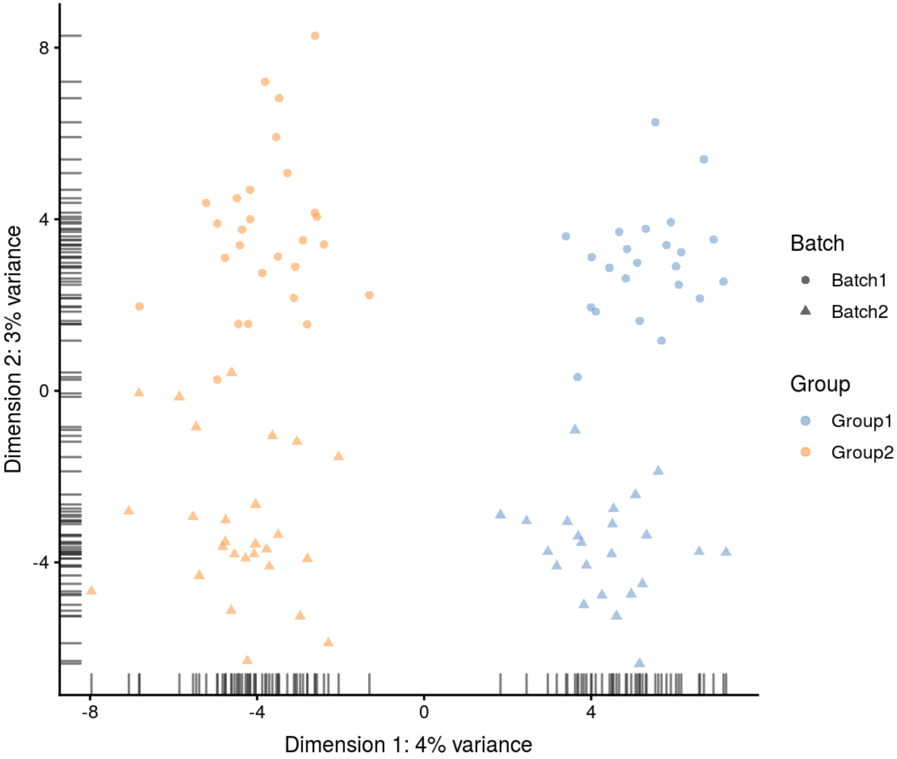
```
sim.groups <- splatSimulate(batchCells = c(50, 50), group.prob = c(0.5, 0.5
),
                            method = "groups", verbose = FALSE)
sim.groups <- normalise(sim.groups)
```

```
## Warning: 'normalise' is deprecated.
## Use 'normalize' instead.
## See help("Deprecated")
```

```
## Warning in .local(object, ...): using library sizes as size factors
```

```
plotPCA(sim.groups, shape_by = "Batch", colour_by = "Group")
```

```
## Warning: 'add_ticks' is deprecated.
## Use '+ geom_rug(...)' instead.
```



Here we see that the effects of the group (first component) are stronger than the batch effects (second component) but by adjusting the parameters we could made the batch effects dominate.

6.4 Convenience functions

Each of the Splatter simulation methods has it's own convenience function. To simulate a single population use `splatSimulateSingle()` (equivalent to `splatSimulate(method = "single")`), to simulate grops use `splatSimulateGroups()` (equivalent to `splatSimulate(method = "groups")`) or to simulate paths use `splatSimulatePaths()` (equivalent to `splatSimulate(method = "paths")`).

7 Other simulations

As well as it's own Splat simulation method the Splatter package contains implementations of other single-cell RNA-seq simulations that have been published or wrappers around simulations included in other packages. To see all the available simulations run the `listSims()` function:

```
listSims()
```

```
## Splatter currently contains 13 simulations
##
## Splat (splat)
## DOI: 10.1186/s13059-017-1305-0    GitHub: Oshlack/splatter
## The Splat simulation generates means from a gamma distribution, adjusts t
hem for BCV and generates counts from a gamma-poisson. Dropout and batch eff
ects can be optionally added.
##
## Splat Single (splatSingle)
## DOI: 10.1186/s13059-017-1305-0    GitHub: Oshlack/splatter
## The Splat simulation with a single population.
##
## Splat Groups (splatGroups)
## DOI: 10.1186/s13059-017-1305-0    GitHub: Oshlack/splatter
## The Splat simulation with multiple groups. Each group can have it's own d
ifferential expression probability and fold change distribution.
##
## Splat Paths (splatPaths)
## DOI: 10.1186/s13059-017-1305-0    GitHub: Oshlack/splatter
## The Splat simulation with differentiation paths. Each path can have it's
own length, skew and probability. Genes can change in non-linear ways.
##
## Simple (simple)
## DOI: 10.1186/s13059-017-1305-0    GitHub: Oshlack/splatter
## A simple simulation with gamma means and negative binomial counts.
##
## Lun (lun)
## DOI: 10.1186/s13059-016-0947-7    GitHub: MarioniLab/Deconvolution2016
## Gamma distributed means and negative binomial counts. Cells are given a s
ize factor and differential expression can be simulated with fixed fold chan
ges.
##
## Lun 2 (lun2)
## DOI: 10.1093/biostatistics/kxw055     GitHub: MarioniLab/PlateEffects2016
## Negative binomial counts where the means and dispersions have been sample
d from a real dataset. The core feature of the Lun 2 simulation is the addit
ion of plate effects. Differential expression can be added between two group
s of plates and optionally a zero-inflated negative-binomial can be used.
##
## scDD (scDD)
## DOI: 10.1186/s13059-016-1077-y    GitHub: kdkorthauer/scDD
## The scDD simulation samples a given dataset and can simulate differential
ly expressed and differentially distributed genes between two conditions.
##
## BASiCS (BASiCS)
## DOI: 10.1371/journal.pcbi.1004333     GitHub: catavallejos/BASiCS
## The BASiCS simulation is based on a bayesian model used to deconvolve bio
logical and technical variation and includes spike-ins and batch effects.
##
## mfa (mfa)
## DOI: 10.12688/wellcomeopenres.11087.1     GitHub: kieranrcampbell/mfa
## The mfa simulation produces a bifurcating pseudotime trajectory. This can
optionally include genes with transient changes in expression and added drop
out.
##
## PhenoPath (pheno)
## DOI: 10.1101/159913   GitHub: kieranrcampbell/phenopath
## The PhenoPath simulation produces a pseudotime trajectory with different
types of genes.
##
## ZINB-WaVE (zinb)
## DOI: 10.1101/125112   GitHub: drisso/zinbwave
## The ZINB-WaVE simulation simulates counts from a sophisticated zero-infla
ted negative-binomial distribution including cell and gene-level covariates.
##
## SparseDC (sparseDC)
## DOI: 10.1093/nar/gkx1113      GitHub: cran/SparseDC
## The SparseDC simulation simulates a set of clusters across two condition
s, where some clusters may be present in only one condition.
```

(or more conveniently for the vignette as a table)

```
knitr::kable(listSims(print = FALSE))
```

Name	Prefix	DOI	GitHub	Description
Splat	splat	10.1186/s13059-017-1305-0	Oshlack/splatter	The Splat simulation generates means from a gamma distribution, adjusts them for BCV and generates counts from a gamma-poisson. Dropout and batch effects can be optionally added.
Splat Single	splatSingle	10.1186/s13059-017-1305-0	Oshlack/splatter	The Splat simulation with a single population.
Splat Groups	splatGroups	10.1186/s13059-017-1305-0	Oshlack/splatter	The Splat simulation with multiple groups. Each group can have it’s own differential expression probability and fold change distribution.
Splat Paths	splatPaths	10.1186/s13059-017-1305-0	Oshlack/splatter	The Splat simulation with differentiation paths. Each path can have it’s own length, skew and probability. Genes can change in non-linear ways.

Name	Prefix	DOI	GitHub	Description
Simple	simple	10.1186/s13059-017-1305-0	Oshlack/splatter	A simple simulation with gamma means and negative binomial counts.
Lun	lun	10.1186/s13059-016-0947-7	MarioniLab/Deconvolution2016	Gamma distributed means and negative binomial counts. Cells are given a size factor and differential expression can be simulated with fixed fold changes.
Lun 2	lun2	10.1093/biostatistics/kxw055	MarioniLab/PlateEffects2016	Negative binomial counts where the means and dispersions have been sampled from a real dataset. The core feature of the Lun 2 simulation is the addition of plate effects. Differential expression can be added between two groups of plates and optionally a zero-inflated negative-binomial can be used.

Name	Prefix	DOI	GitHub	Description
scDD	scDD	10.1186/s13059-016-1077-y	kdkorthauer/scDD	The scDD simulation samples a given dataset and can simulate differentially expressed and differentially distributed genes between two conditions.
BASiCS	BASiCS	10.1371/journal.pcbi.1004333	catavallejos/BASiCS	The BASiCS simulation is based on a bayesian model used to deconvolve biological and technical variation and includes spike-ins and batch effects.
mfa	mfa	10.12688/wellcomeopenres.11087.1	kieranrcampbell/mfa	The mfa simulation produces a bifurcating pseudotime trajectory. This can optionally include genes with transient changes in expression and added dropout.
PhenoPath	pheno	10.1101/159913	kieranrcampbell/phenopath	The PhenoPath simulation produces a pseudotime trajectory with different types of genes.

Name	Prefix	DOI	GitHub	Description
ZINB-WaVE	zinb	10.1101/125112	drisso/zinbwave	The ZINB-WaVE simulation simulates counts from a sophisticated zero-inflated negative-binomial distribution including cell and gene-level covariates.
SparseDC	sparseDC	10.1093/nar/gkx1113	cran/SparseDC	The SparseDC simulation simulates a set of clusters across two conditions, where some clusters may be present in only one condition.

Each simulation has it's own prefix which gives the name of the functions associated with that simulation. For example the prefix for the simple simulation is `simple` so it would store it's parameters in a `SimpleParams` object that can be created using `newSimpleParams()` or estimated from real data using `simpleEstimate()` . To simulate data using that simulation you would use `simpleSimulate()` . Each simulation returns a `SingleCellExperiment` object with intermediate values similar to that returned by `splatSimulate()` . For more detailed information on each simulation see the appropriate help page (eg. `?simpleSimulate` for information on how the simple simulation works or `? lun2Estimate` for details of how the Lun 2 simulation estimates parameters) or refer to the appropriate paper or package.

8 Other expression values

Splatter is designed to simulate count data but some analysis methods expect other expression values, particularly length-normalised values such as TPM or FPKM. The `scater` package has functions for adding these values to a `SingleCellExperiment` object but they require a length for each gene. The `addGeneLengths` function can be used to simulate these lengths:

```
sim <- simpleSimulate(verbose = FALSE)
sim <- addGeneLengths(sim)
head(rowData(sim))
```

```
## DataFrame with 6 rows and 3 columns
##             Gene             GeneMean    Length
##         <factor>            <numeric> <numeric>
## Gene1    Gene1   0.0102807805741105      2013
## Gene2    Gene2      2.8999730077771      5092
## Gene3    Gene3    0.573331819951632     25131
## Gene4    Gene4     2.07904650149391      2217
## Gene5    Gene5 1.59070618087396e-05      1534
## Gene6    Gene6      0.2919204944368      2313
```

We can then use `scater` to calculate TPM:

```
tpm(sim) <- calculateTPM(sim, rowData(sim)$Length)
tpm(sim)[1:5, 1:5]
```

```
##             Cell1    Cell2    Cell3    Cell4    Cell5
## Gene1   0.00000  0.00000  0.00000   0.0000   0.0000
## Gene2  93.81678 62.23436 62.05361 155.1567 157.4989
## Gene3  12.67266  6.30491 12.57320   0.0000   0.0000
## Gene4 287.30421  0.00000  0.00000 142.5454 434.0916
## Gene5   0.00000  0.00000  0.00000   0.0000   0.0000
```

The default method used by `addGeneLengths` to simulate lengths is to generate values from a log-normal distribution which are then rounded to give an integer length. The parameters for this distribution are based on human protein coding genes but can be adjusted if needed (for example for other species). Alternatively lengths can be sampled from a provided vector (see `?addGeneLengths` for details and an example).

9 Comparing simulations and real data

One thing you might like to do after simulating data is to compare it to a real dataset, or compare simulations with different parameters or models. Splatter provides a function `compareSCEs` that aims to make these comparisons easier. As the name suggests this function takes a list of `SingleCellExperiment` objects, combines the datasets and produces some plots comparing them. Let's make two small simulations and see how they compare.

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

names(comparison)
```

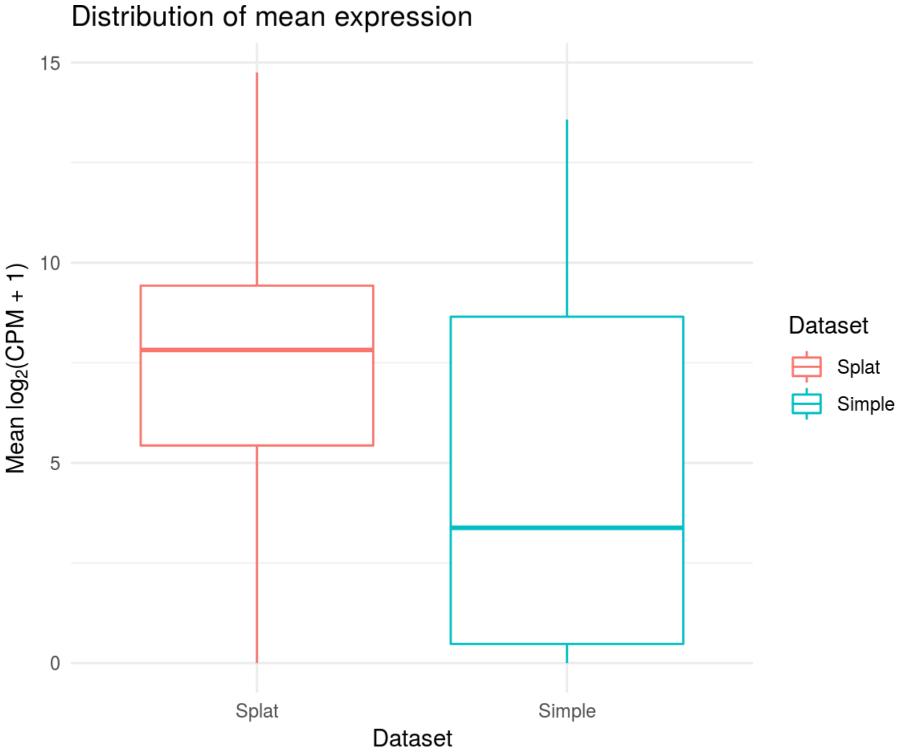
```
## [1] "FeatureData" "PhenoData"   "Plots"
```

```
names(comparison$Plots)
```

```
## [1] "Means"         "Variances"    "MeanVar"       "LibrarySizes"
## [5] "ZerosGene"     "ZerosCell"    "MeanZeros"
```

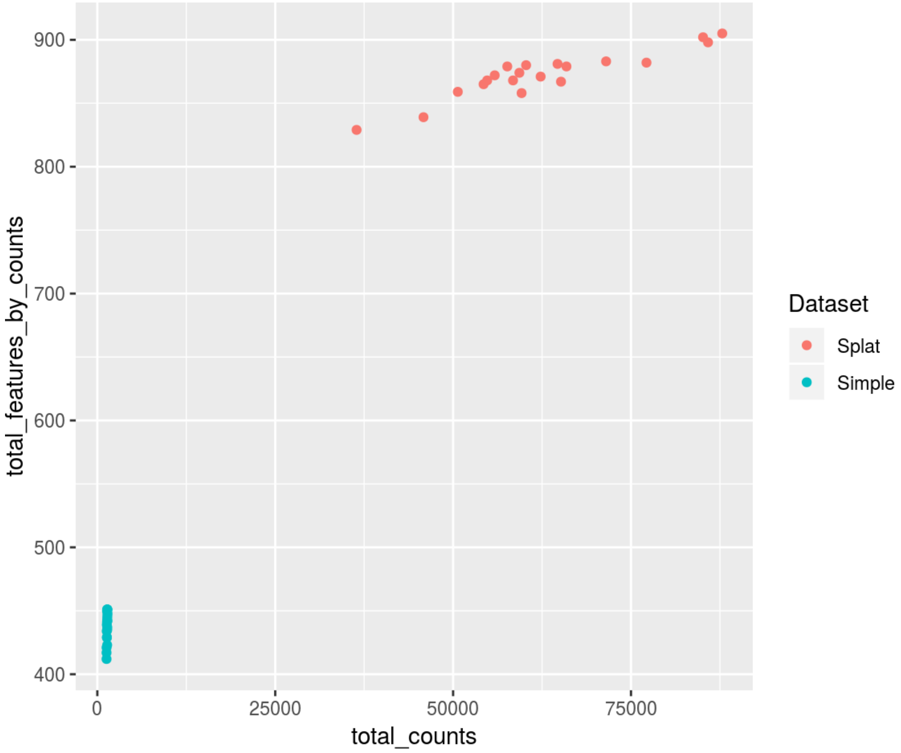
The returned list has three items. The first two are the combined datasets by gene (`FeatureData`) and by cell (`PhenoData`) and the third contains some comparison plots (produced using `ggplot2`), for example a plot of the distribution of means:

```
comparison$Plots$Means
```



These are only a few of the plots you might want to consider but it should be easy to make more using the returned data. For example, we could plot the number of expressed genes against the library size:

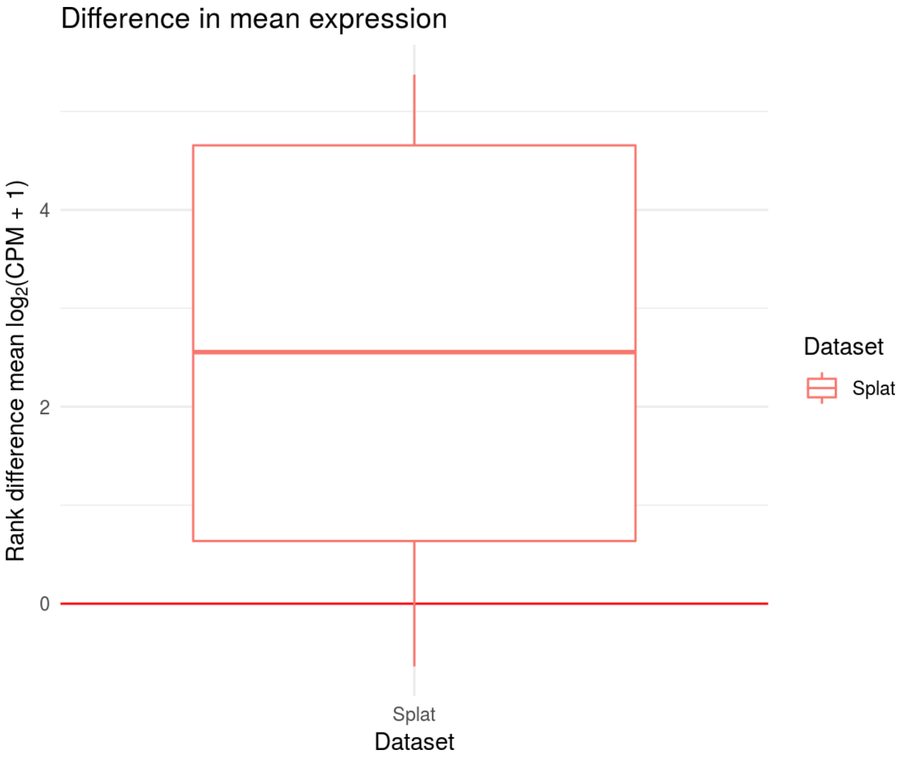
```
library("ggplot2")
ggplot(comparison$PhenoData,
       aes(x = total_counts, y = total_features_by_counts, colour = Datase
t)) +
    geom_point()
```



9.1 Comparing differences

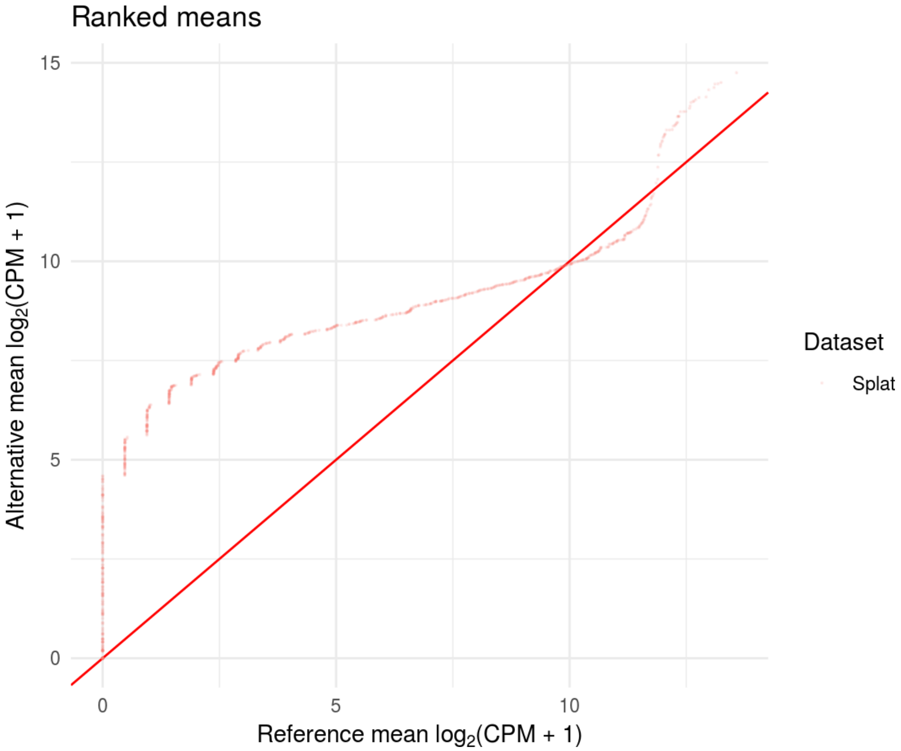
Sometimes instead of visually comparing datasets it may be more interesting to look at the differences between them. We can do this using the `diffSCEs` function. Similar to `compareSCEs` this function takes a list of `SingleCellExperiment` objects but now we also specify one to be a reference. A series of similar plots are returned but instead of showing the overall distributions they demonstrate differences from the reference.

```
difference <- diffSCEs(list(Splat = sim1, Simple = sim2), ref = "Simple")
difference$Plots$Means
```



We also get a series of Quantile-Quantile plot that can be used to compare distributions.

```
difference$QQPlots$Means
```



9.2 Making panels

Each of these comparisons makes several plots which can be a lot to look at. To make this easier, or to produce figures for publications, you can make use of the functions `makeCompPanel` , `makeDiffPanel` and `makeOverallPanel` .

These functions combine the plots into a single panel using the `cowplot` package. The panels can be quite large and hard to view (for example in RStudio's plot viewer) so it can be better to output the panels and view them separately. Luckily `cowplot` provides a convenient function for saving the images. Here are some suggested parameters for outputting each of the panels:

```
# This code is just an example and is not run
panel <- makeCompPanel(comparison)
cowplot::save_plot("comp_panel.png", panel, nrow = 4, ncol = 3)

panel <- makeDiffPanel(difference)
cowplot::save_plot("diff_panel.png", panel, nrow = 3, ncol = 5)

panel <- makeOverallPanel(comparison, difference)
cowplot::save_plot("overall_panel.png", panel, ncol = 4, nrow = 7)
```

10 Citing Splatter

If you use Splatter in your work please cite our paper:

```
citation("splatter")
```

```
##
##   Zappia L, Phipson B, Oshlack A. Splatter: Simulation of
##   single-cell RNA sequencing data. Genome Biology. 2017;
##   doi:10.1186/s13059-017-1305-0
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     author = {Luke Zappia and Belinda Phipson and Alicia Oshlack},
##     title = {Splatter: simulation of single-cell RNA sequencing data},
##     journal = {Genome Biology},
##     year = {2017},
##     url = {http://dx.doi.org/10.1186/s13059-017-1305-0},
##     doi = {10.1186/s13059-017-1305-0},
##   }
```

Session information

```
sessionInfo()
```

```
## R version 3.5.1 Patched (2018-07-12 r74967)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 16.04.5 LTS
##
## Matrix products: default
## BLAS: /home/biocbuild/bbs-3.8-bioc/R/lib/libRblas.so
## LAPACK: /home/biocbuild/bbs-3.8-bioc/R/lib/libRlapack.so
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] scater_1.10.0               ggplot2_3.1.0
##  [3] splatter_1.6.1              SingleCellExperiment_1.4.0
##  [5] SummarizedExperiment_1.12.0 DelayedArray_0.8.0
##  [7] BiocParallel_1.16.2         matrixStats_0.54.0
##  [9] Biobase_2.42.0              GenomicRanges_1.34.0
## [11] GenomeInfoDb_1.18.1         IRanges_2.16.0
## [13] S4Vectors_0.20.1            BiocGenerics_0.28.0
## [15] BiocStyle_2.10.0
##
## loaded via a namespace (and not attached):
##  [1] viridis_0.5.1            edgeR_3.24.1
##  [3] viridisLite_0.3.0        splines_3.5.1
##  [5] lsei_1.2-0               DelayedMatrixStats_1.4.0
##  [7] assertthat_0.2.0         highr_0.7
##  [9] sp_1.3-1                 BiocManager_1.30.4
## [11] GenomeInfoDbData_1.2.0   vipor_0.4.5
## [13] yaml_2.2.0               pillar_1.3.0
## [15] backports_1.1.2          lattice_0.20-38
## [17] limma_3.38.3             glue_1.3.0
## [19] digest_0.6.18            XVector_0.22.0
## [21] checkmate_1.8.5          colorspace_1.3-2
## [23] cowplot_0.9.3            htmltools_0.3.6
## [25] Matrix_1.2-15            plyr_1.8.4
## [27] pkgconfig_2.0.2          bookdown_0.8
## [29] zlibbioc_1.28.0          purrr_0.2.5
## [31] scales_1.0.0             HDF5Array_1.10.1
## [33] tibble_1.4.2             withr_2.1.2
## [35] lazyeval_0.2.1           survival_2.43-3
## [37] magrittr_1.5             crayon_1.3.4
## [39] evaluate_0.12            MASS_7.3-51.1
## [41] beeswarm_0.2.3           tools_3.5.1
## [43] fitdistrplus_1.0-11      stringr_1.3.1
## [45] Rhdf5lib_1.4.2           munsell_0.5.0
## [47] locfit_1.5-9.1           bindrcpp_0.2.2
## [49] akima_0.6-2              compiler_3.5.1
## [51] rlang_0.3.0.1            rhdf5_2.26.0
## [53] grid_3.5.1               RCurl_1.95-4.11
## [55] labeling_0.3             bitops_1.0-6
## [57] rmarkdown_1.10           npsurv_0.4-0
## [59] gtable_0.2.0             reshape2_1.4.3
## [61] R6_2.3.0                 gridExtra_2.3
## [63] knitr_1.20               dplyr_0.7.8
## [65] bindr_0.1.1              rprojroot_1.3-2
## [67] stringi_1.2.4            ggbeeswarm_0.6.0
## [69] Rcpp_1.0.0               tidyselect_0.2.5
## [71] xfun_0.4
```