Splat simulation parameters

Last updated: 18 April 2019

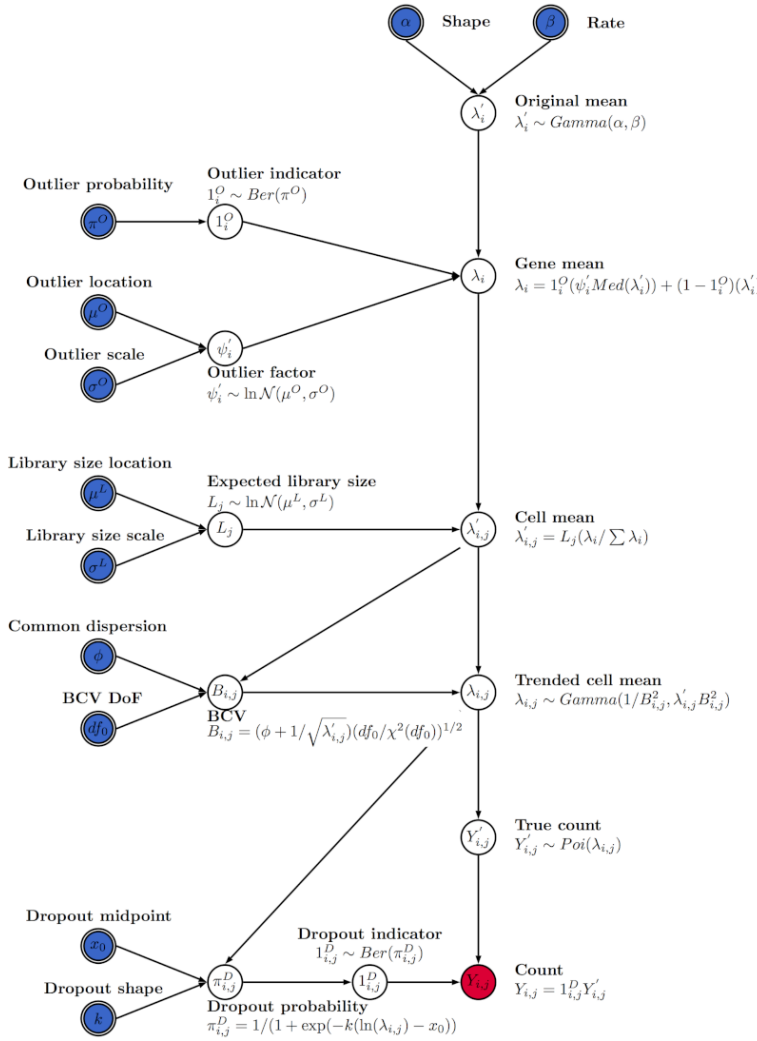
This vignette describes the Splat simulation model and the parameters it uses in more detail.

```
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, clusterEva
lQ,
#>     clusterExport, clusterMap, parApply, parCapply, parLa
pply,
#>     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, a
ppend,
#>     as.data.frame, basename, cbind, colnames, dirname, d
o.call,
#>     duplicated, eval, evalq, get, grep, grepl, intersect,
#>     is.unsorted, lapply, mapply, match, mget, order, past
e, pmax,
#>     pmax.int, pmin, pmin.int, rank, rbind, rownames, sapp
ly,
#>     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("pk
gname")'.
#> 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:matrixStat
s':
#>
```

```
#>     colMaxs, colMins, colRanges, rowMaxs, rowMins, rowRan
ges
#> The following objects are masked from 'package:base':
#>
#>     aperm, apply, rowsum
#> Registered S3 methods overwritten by 'ggplot2':
#>   method         from
#>   [.quosures     rlang
#>   c.quosures     rlang
#>   print.quosures rlang
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
library("ggplot2")
```

1 The base Splat model

This figure, taken from the Splatter publication, describes the core of the Splat simulation model.



Splat simulation model

The Splat simulation uses a hierarchical probabilistic where different aspects of a dataset are generated from appropriate statistical distributions. The first stage generates a mean expression level for each gene. These are originally chosen from a Gamma distribution. For some genes that are selected to be outliers with high expression a factor is generated from a log-normal distribution. These factors are then multiplied by the median gene mean to create new means for those genes.

The next stage incorporates variation in the counts per cell. An expected library size (total counts) is chosen for each cell from a log-normal distribution. The library sizes are then used to scale the gene means for each cell, resulting in a range a counts per cell in the simulated dataset. The gene means are then further adjusted to enforce a relationship between the mean expression level and the variability.

The final cell by gene matrix of gene means is then used to generate a count matrix using a Poisson distribution. The result is a synthetic dataset consisting of counts from a Gamma-Poisson (or negative-binomial) distribution. An additional optional step can be used to replicate a “dropout” effect. A probability of dropout is generated using a logistic function based on the underlying mean expression level. A Bernoulli distribution is then used to create a dropout matrix which sets some of the generated counts to zero.

The model described here will generate a single population of cells but the Splat simulation has been designed to be as flexible as possible and can create scenarios including multiple groups of cells (cell types), continuous paths between cell types and multiple experimental batches. The parameters used to create these types of simulations and how they interact with the model are described below.

2 Splat simulation parameters

Within Splatter the parameters for the Splat simulation model are held in the `SplatParams` object. Let’s create one of these objects and see what it looks like.

```
params <- newSplatParams()
params
#> A Params object of class SplatParams
#> Parameters can be (estimable) or [not estimable], 'Defaul
t' or  'NOT DEFAULT'
#>
#> Global:
#> (Genes)  (Cells)   [Seed]
#>   10000      100   138572
#>
#> 28 additional parameters
#>
#> Batches:
#>     [Batches]  [Batch Cells]     [Location]        [Scal
e]
#>             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]        [Scal
e]
#>           0.1            0.5            0.1            0.
4
#>
#> BCV:
#> (Common Disp)          (DoF)
#>           0.1             60
#>
#> Dropout:
#>     [Type]  (Midpoint)     (Shape)
#>       none           0          -1
#>
#> Paths:
#>         [From]         [Steps]          [Skew]    [Non-li
near]
#>              0             100             0.5
0.1
#> [Sigma Factor]
#>            0.8
```

Like all the parameter objects in Splatter printing this object displays all the parameters required for this simulation. As we haven't set any of the parameters the default values are shown but if we were to change any of them they would be highlighted. We can also see which parameters can be

estimated by the Splat estimation procedure and which can't. The default values have been chosen to be fairly realistic but it is recommended that estimation is used to get a simulation that is more like the data you are interested in. Parameters can be modified by setting them in the `SplatParams` object or by providing them directly to the simulation function.

The rest of this section provides details of all this parameters and explains how they can be used with examples.

2.1 Global parameters

These parameters are used in every simulation model and control global features of the dataset produced.

2.1.1 `nGenes` - Number of genes

The number of genes to simulate.

```
# Set the number of genes to 1000
params <- setParam(params, "nGenes", 1000)

sim <- splatSimulate(params, verbose = FALSE)
dim(sim)
#> [1] 1000  100
```

2.1.2 `nCells` - Number of cells

The number of genes to simulate. In the Splat simulation this cannot be set directly but must be controlled using the `batchCells` parameter.

2.1.3 `seed` - Random seed

Seed to use for generating random numbers including selecting values from distributions. By changing this value multiple simulated datasets with the same parameters can be produced. Simulations produced using the same set of parameters and random seed should be identical but there may be differences between operating systems, software versions etc.

2.2 Batch parameters

These parameters control experimental batches in the simulated dataset. The overall effect of how batch effects are included in the model is similar to technical replicates (i.e. the same biological sample sequenced multiple times). This means that the underlying structure is consistent between batches but a global technical signature is added may separate them.

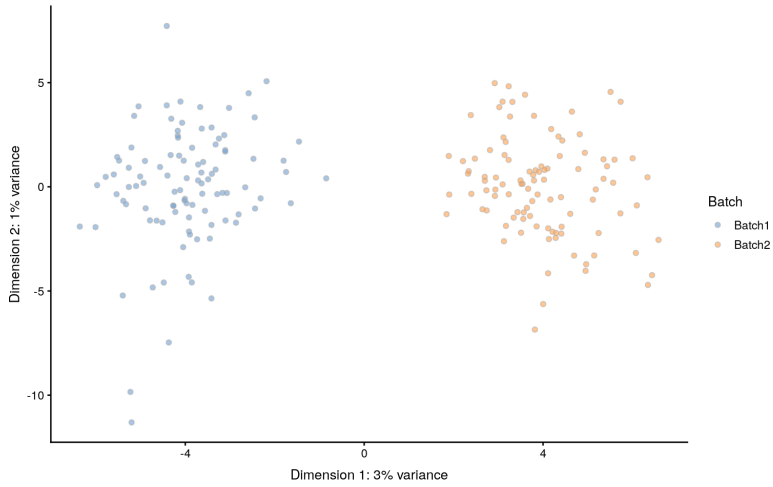
2.2.1 `nBatches` - Number of batches

The number of batches in the simulation. This cannot be set directly but is controlled by setting `batchCells` .

2.2.2 `batchCells` - Cells per batch

A vector specifying the number of cells in each batch. The number of batches (`nBatches`) is equal to the length of the vector and the number of cells (`nCells`) is equal to the sum.

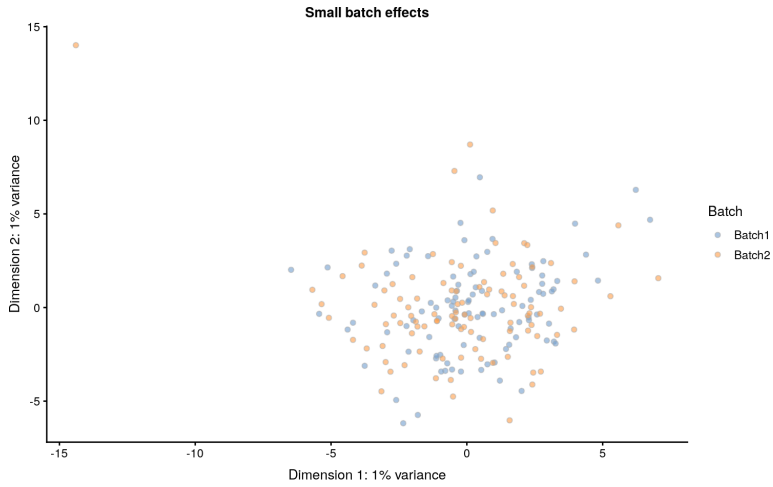
```
# Simulation with two batches of 100 cells
sim <- splatSimulate(params, batchCells = c(100, 100), verbo
se = FALSE)

# PCA plot using scater
sim <- normalize(sim)
#> Warning in .local(object, ...): using library sizes as si
ze factors
plotPCA(sim, colour_by = "Batch")
```



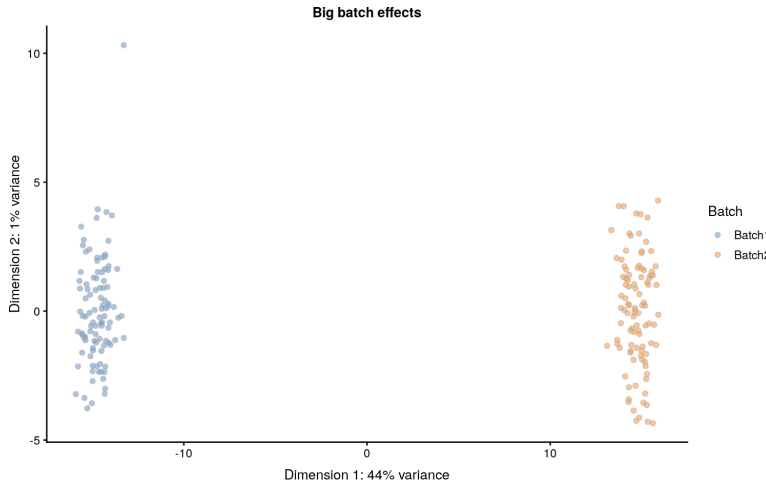
2.2.3 `batch.facLoc` - Batch factor location and `batch.facScale` - Batch factor scale

Batches are specified by generating a small scaling factor for each gene in each batch from a log-normal distribution. These factors are then applied to the underlying gene means in each batch. Modifying these parameters affects how different the batches are from each other by generating bigger or smaller factors.

```
# Simulation with small batch effects
sim1 <- splatSimulate(params, batchCells = c(100, 100),
                      batch.facLoc = 0.001, batch.facScale =
0.001,
                      verbose = FALSE)
sim1 <- normalize(sim1)
#> Warning in .local(object, ...): using library sizes as si
ze factors
plotPCA(sim1, colour_by = "Batch") + ggtitle("Small batch ef
fects")
```



```
# Simulation with big batch effects
sim2 <- splatSimulate(params, batchCells = c(100, 100),
                      batch.facLoc = 0.5, batch.facScale =
0.5,
                      verbose = FALSE)
sim2 <- normalize(sim2)
#> Warning in .local(object, ...): using library sizes as si
ze factors
plotPCA(sim2, colour_by = "Batch") + ggtitle("Big batch effe
cts")
```



2.3 Mean parameters

These parameters control the distribution that is used to generate the underlying original gene means.

2.3.1 `mean.shape` - Mean shape and `mean.rate` - Mean rate

These parameters control the Gamma distribution that gene means are drawn from. The relationship between shape and rate can be complex and it is often better to use values estimated from a real dataset than to try and manually set them. Although these parameters control the base gene means the means in the final simulation will depend upon other parts of the model, particularly the simulated total counts per cell (library sizes).

2.4 Library size parameters

These parameters control the expected number of counts for each cell. Note that because of sampling the actual counts per cell in the final simulation may be different. Turning on the dropout effect will also effect this. We use the term “library size” here for consistency but expected total counts would be more appropriate.

2.4.1 `lib.loc` - Library size location and `lib.scale` - Library size scale

These parameters control the shape of the distribution that is used to generate library sizes for each cell. Increasing `lib.loc` will lead to more counts per cell and increasing `lib.scale` will result in more variability in the counts per cell.

2.4.2 `lib.norm` - Library size distribution

The default (and recommended) distribution used for library sizes in the Splat simulation is a log-normal. However, in rare cases a normal distribution might be more appropriate. Setting `lib.norm` to `TRUE` will use a normal distribution instead of a log-normal.

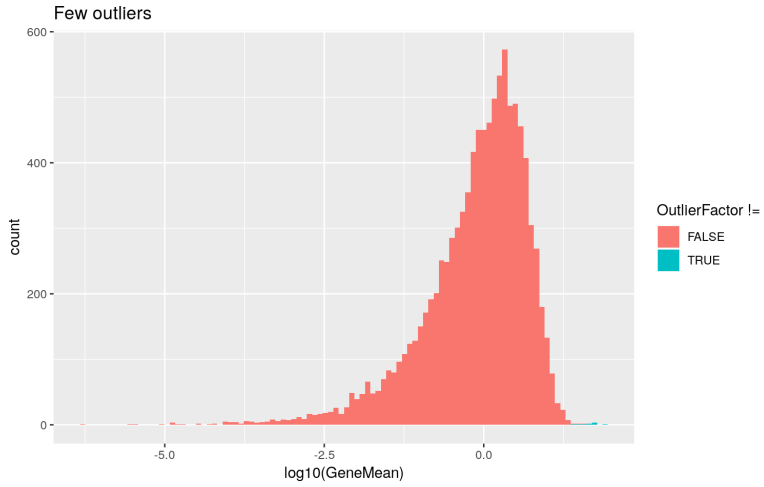
2.5 Expression outlier parameters

When developing the Splat simulation we found that while the Gamma distribution was generally a good match for gene means for some datasets it did not properly capture highly expressed genes. For this reason we added expression outliers to the Splat model.

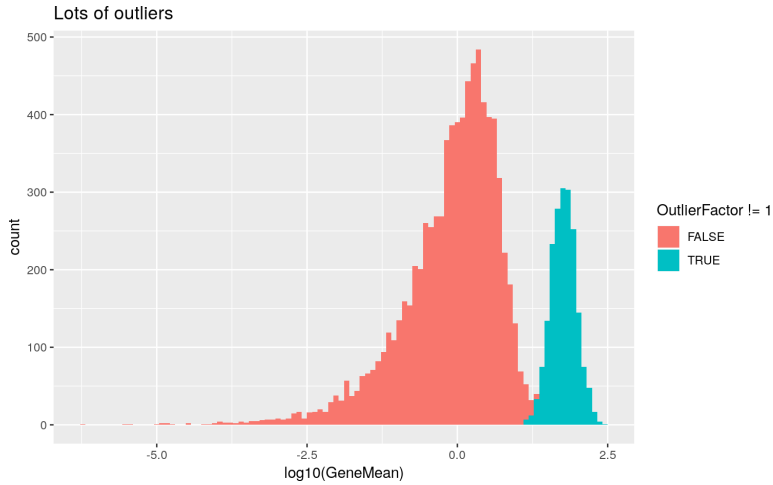
2.5.1 `out.prob` - Expression outlier probability

This parameter controls the probability that genes will be selected to be expression outliers. Higher values will results in more outlier genes.

```
# Few outliers
sim1 <- splatSimulate(out.prob = 0.001, verbose = FALSE)
ggplot(as.data.frame(rowData(sim1)),
       aes(x = log10(GeneMean), fill = OutlierFactor != 1))
 +
    geom_histogram(bins = 100) +
    ggtitle("Few outliers")
```




```
# Lots of outliers
sim2 <- splatSimulate(out.prob = 0.2, verbose = FALSE)
ggplot(as.data.frame(rowData(sim2)),
       aes(x = log10(GeneMean), fill = OutlierFactor != 1))
 +
    geom_histogram(bins = 100) +
    ggtitle("Lots of outliers")
```



2.5.2 `out.facLoc` - Expression outlier factor location and `out.facScale` - Expression outlier factor scale

The expression outlier factors are drawn from a log-normal distribution controlled by these parameters. The generated factors are applied to the median mean expression rather than the existing mean for the selected genes. This is to be consistent with the estimation procedure for these factors and to make sure that the final means are outliers. For example to avoid the situation where a factor is applied to a lowly expressed gene, making it just moderately expressed rather than an expression outlier.

2.5.3 Group parameters

Up until this stage of the simulation only a single population of cells is considered but we often want to simulate datasets with multiple kinds of cells. We do this by assigning cells to groups.

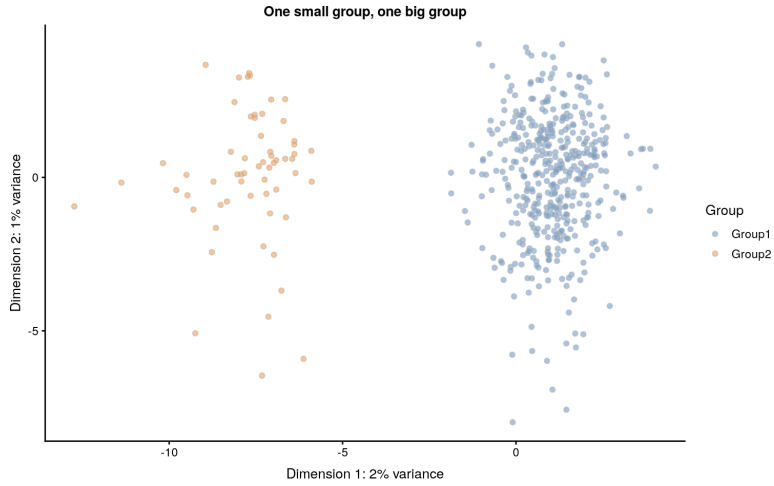
2.5.4 `nGroups` - Number of groups

The number of groups to simulate. This parameter cannot be set directly and is controlled using `group.prob` .

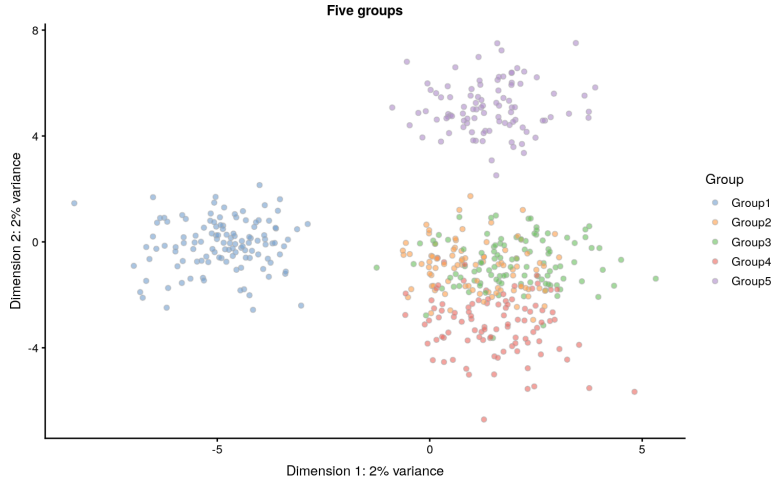
2.5.5 `group.prob` - Group probabilities

A vector giving the probability that cells will be assigned to groups. The length of the vector gives the number of groups (`nGroups`) and the probabilities must sum to 1. Adjusting the number and relative values of the probabilities changes the number and relative sizes of the groups. To simulate groups we also need to use the `splatSimulateGroups` function or set `method = "groups"` .

```
params.groups <- newSplatParams(batchCells = 500, nGenes = 1
000)

# One small group, one big group
sim1 <- splatSimulateGroups(params.groups, group.prob = c(0.
9, 0.1),
                            verbose = FALSE)
sim1 <- normalize(sim1)
#> Warning in .local(object, ...): using library sizes as si
ze factors
plotPCA(sim1, colour_by = "Group") + ggtitle("One small grou
p, one big group")
```




```
# Five groups
sim2 <- splatSimulateGroups(params.groups,
                            group.prob = c(0.2, 0.2, 0.2, 0.
2, 0.2),
                            verbose = FALSE)
sim2 <- normalize(sim2)
#> Warning in .local(object, ...): using library sizes as si
ze factors
plotPCA(sim2, colour_by = "Group") + ggtitle("Five groups")
```



Note: Once there are more than three or four groups it becomes difficult to properly view them in PCA space. We use PCA here for simplicity but generally a non-linear dimensionality reduction such as t-SNE or UMAP is a more useful way to visualise the groups.

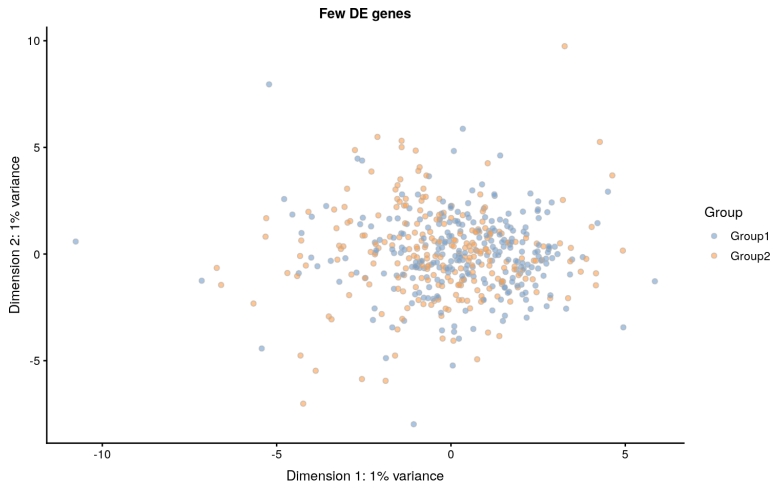
2.5.6 Differential expression parameters

Different groups are created by modifying the base expression levels of selected genes. The process for doing this is to simulate differential expression (DE) between each group and a fictional base cell. Altering the differential expression parameters controls how similar groups are to each other.

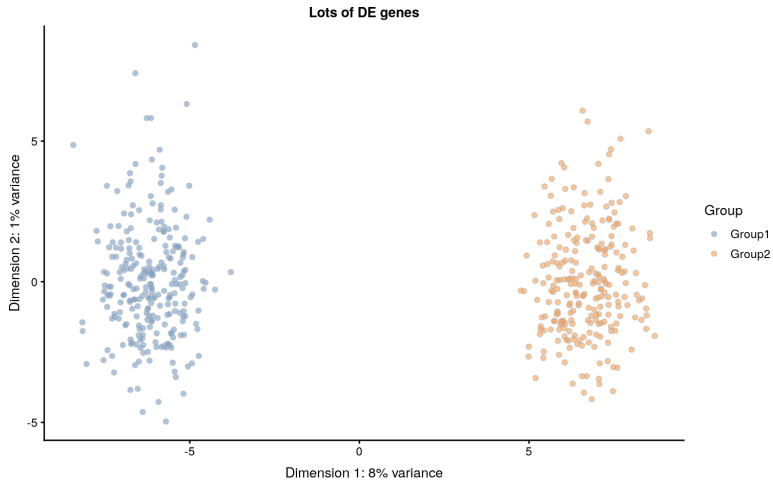
2.5.7 `de.prob` - DE probability

This parameter controls the probability that a gene will be selected to be differentially expressed.

```
# Few DE genes
sim1 <- splatSimulateGroups(params.groups, group.prob = c(0.
5, 0.5),
                            de.prob = 0.01, verbose = FALSE)
sim1 <- normalize(sim1)
#> Warning in .local(object, ...): using library sizes as si
ze factors
plotPCA(sim1, colour_by = "Group") + ggtitle("Few DE genes")
```




```
# Lots of DE genes
sim2 <- splatSimulateGroups(params.groups, group.prob = c(0.
5, 0.5),
                            de.prob = 0.3, verbose = FALSE)
sim2 <- normalize(sim2)
#> Warning in .local(object, ...): using library sizes as si
ze factors
plotPCA(sim2, colour_by = "Group") + ggtitle("Lots of DE gen
es")
```



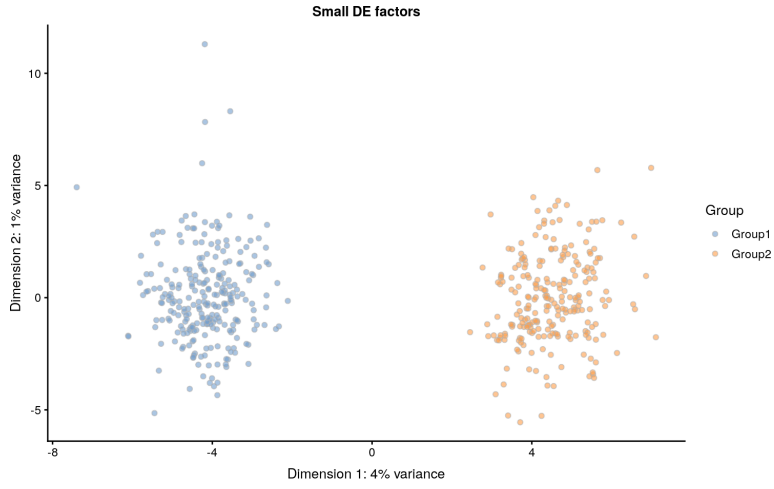
2.5.8 `de.downProb` - Down-regulation probability

A selected DE gene can be either down-regulated (has a factor less than one) or up-regulated (has a factor greater than one). This parameter controls the probability that a selected gene will be down-regulated.

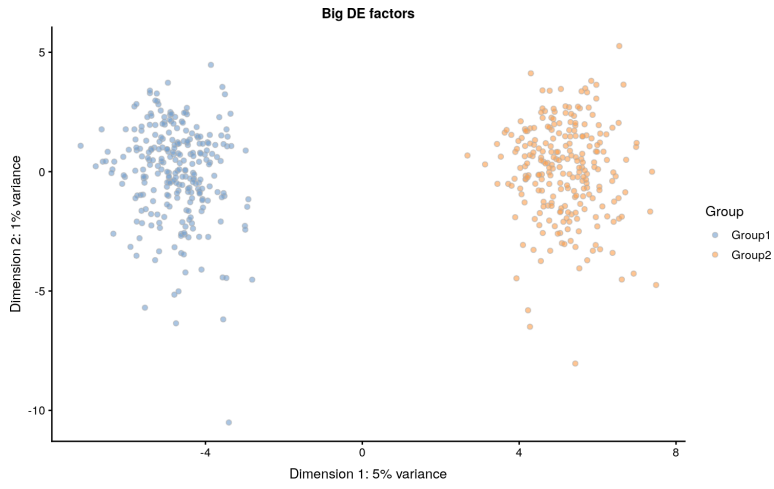
2.5.9 `de.facLoc` - DE factor location and `de.facScale` - DE factor scale

Differential expression factors are produced from a log-normal distribution in a similar way to batch effect factors and expression outlier factors. Changing these parameters can result in more or less extreme differences between groups.

```
# Small DE factors
sim1 <- splatSimulateGroups(params.groups, group.prob = c(0.
5, 0.5),
                            de.facLoc = 0.01, verbose = FALS
E)
sim1 <- normalize(sim1)
#> Warning in .local(object, ...): using library sizes as si
ze factors
plotPCA(sim1, colour_by = "Group") + ggtitle("Small DE facto
rs")
```




```
# Big DE factors
sim2 <- splatSimulateGroups(params.groups, group.prob = c(0.
5, 0.5),
                            de.facLoc = 0.3, verbose = FALSE
)
sim2 <- normalize(sim2)
#> Warning in .local(object, ...): using library sizes as si
ze factors
plotPCA(sim2, colour_by = "Group") + ggtitle("Big DE factor
s")
```

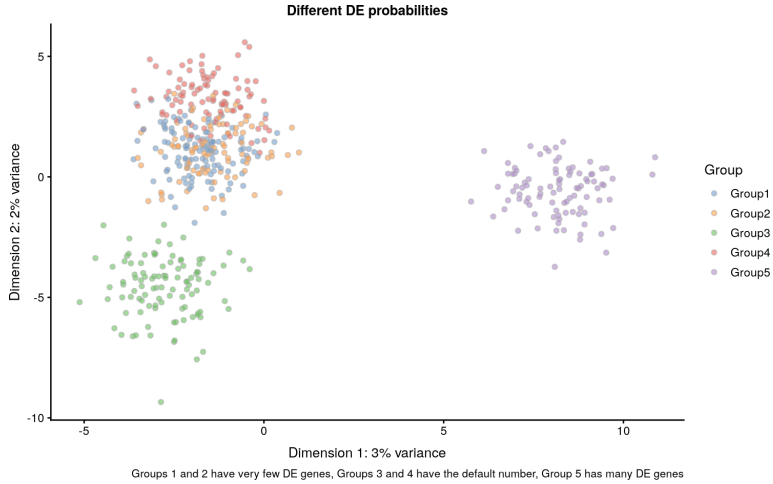


Just looking at the PCA plots this effect seems similar to adjusting `de.prob` but the effect is achieved in a different way. A higher `de.prob` means that more genes are differentially expressed but changing the DE factors changes the level of DE for the same number of genes.

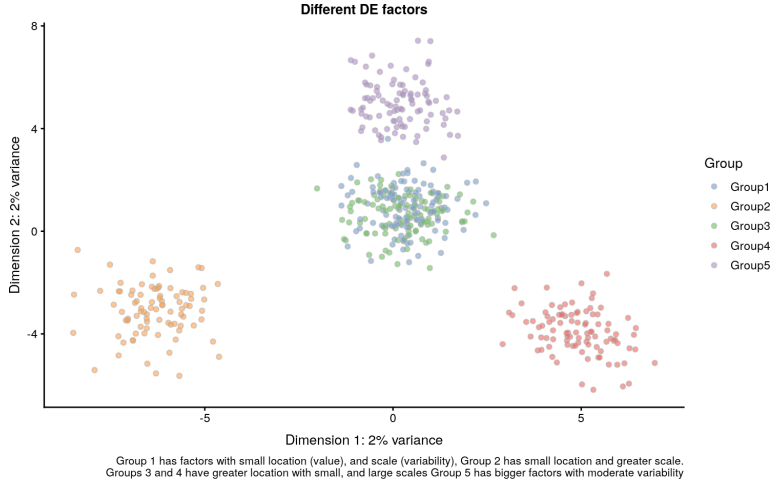
2.5.10 Complex differential expression

Each of the differential expression parameters can be specified for each group by providing a vector of values. These vectors must be the same length as `group.prob`. Specifying parameters as vectors allows more complex simulations where groups are more or less different to each other rather than being equally distinct. Here are some examples of different DE scenarios.

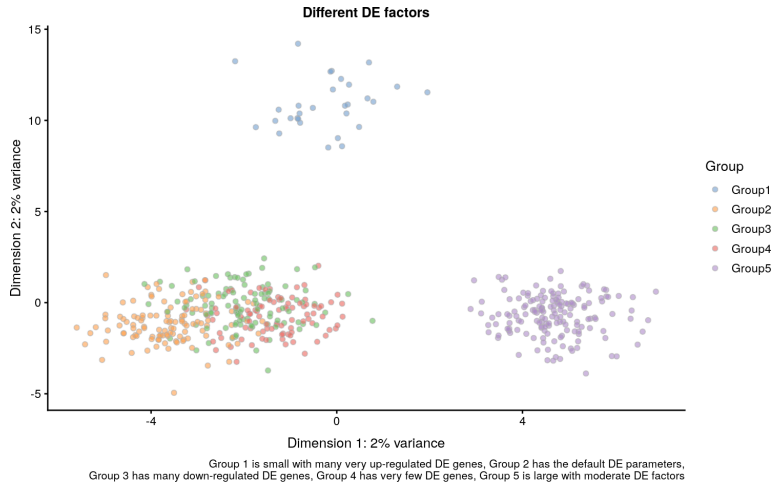
```
# Different DE probs
sim1 <- splatSimulateGroups(params.groups,
                            group.prob = c(0.2, 0.2, 0.2, 0.
2, 0.2),
                            de.prob = c(0.01, 0.01, 0.1, 0.1
, 0.3),
                            verbose = FALSE)
sim1 <- normalize(sim1)
#> Warning in .local(object, ...): using library sizes as si
ze factors
plotPCA(sim1, colour_by = "Group") +
    labs(title = "Different DE probabilities",
         caption = paste("Groups 1 and 2 have very few DE ge
nes,",
                         "Groups 3 and 4 have the default nu
mber,",
                         "Group 5 has many DE genes"))
```



```
# Different DE factors
sim2 <- splatSimulateGroups(params.groups,
                            group.prob = c(0.2, 0.2, 0.2, 0.
2, 0.2),
                            de.facLoc = c(0.01, 0.01, 0.1,
0.1, 0.2),
                            de.facScale = c(0.2, 0.5, 0.2,
0.5, 0.4),
                            verbose = FALSE)
sim2 <- normalize(sim2)
#> Warning in .local(object, ...): using library sizes as si
ze factors
plotPCA(sim2, colour_by = "Group") +
    labs(title = "Different DE factors",
         caption = paste("Group 1 has factors with small loc
ation (value),",
                         "and scale (variability),",
                         "Group 2 has small location and gre
ater scale.\n",
                         "Groups 3 and 4 have greater locati
on with small,",
                         "and large scales",
                         "Group 5 has bigger factors with mo
derate variability"))
```



```
# Combination of everything
sim3 <- splatSimulateGroups(params.groups,
                            group.prob = c(0.05, 0.2, 0.2,
0.2, 0.35),
                            de.prob = c(0.3, 0.1, 0.2, 0.01,
0.1),
                            de.downProb = c(0.1, 0.4, 0.9,
0.6, 0.5),
                            de.facLoc = c(0.6, 0.1, 0.1, 0.0
1, 0.2),
                            de.facScale = c(0.1, 0.4, 0.2,
0.5, 0.4),
                            verbose = FALSE)
sim3 <- normalize(sim3)
#> Warning in .local(object, ...): using library sizes as si
ze factors
plotPCA(sim3, colour_by = "Group") +
    labs(title = "Different DE factors",
         caption = paste(
             "Group 1 is small with many very up-regulated D
E genes,",
             "Group 2 has the default DE parameters,\n",
             "Group 3 has many down-regulated DE genes,",
             "Group 4 has very few DE genes,",
             "Group 5 is large with moderate DE factors")
         )
```



2.6 Biological Coefficient of Variation (BCV) parameters

The BCV parameters control the variability of the genes in the simulated dataset.

2.6.1 `bcv.common` - Common BCV

The `bcv.common` parameter controls the underlying common variability across all genes in the dataset.

2.6.2 `bcv.df` - BCV Degrees of Freedom

This parameter sets the degrees of freedom used in the BCV inverse chi-squared distribution. Changing this changes the effect of mean expression on the variability of a gene.

2.7 Dropout parameters

These parameters control whether additional dropout is added to increase the number of zeros in the simulated dataset and if it is how that is applied.

2.7.1 `dropout.type` - Dropout type

This parameter determines the kind of dropout effect to simulate. Setting it to `"none"` means no dropout, "`experiment`" is global dropout using the same set of parameters for every cell, `"batch"` uses the same parameters for every cell in the same batch, `"group"` uses the same parameters for every cell in the same group and `"cell"` uses a different set of parameters for every cell.

2.7.2 `dropout.mid` - Dropout mid point and `dropout.shape` - Dropout shape

The probability that a particular count in a particular cell is set to zero is related to the mean expression of that gene in that cell. This relationship is represented using a logistic function with these parameters. The `dropout.mid` parameter control the point at which the probability is equal to 0.5 and the `dropout.shape` controls how it changes with increasing expression. These parameters must be vectors of the appropriate length depending on the selected dropout type.

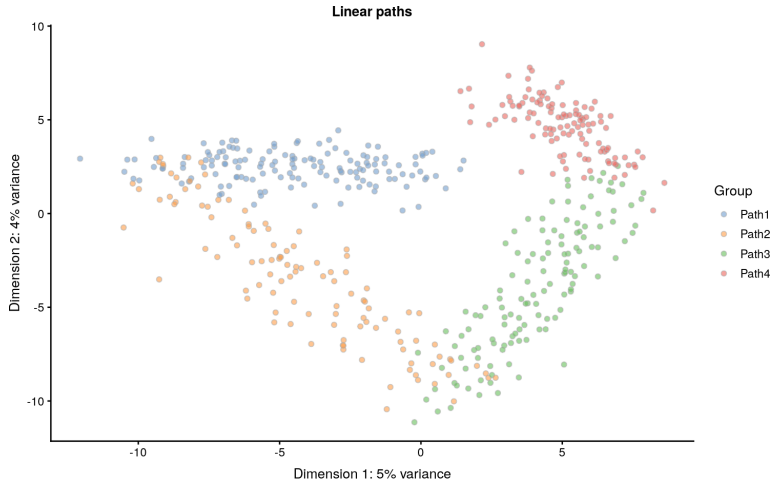
2.8 Path parameters

For many uses simulating groups is sufficient but in some cases it is more appropriate to simulate continuous changes between cell types. The number of paths and the probability of assigning cells to them is still controlled by the `group.prob` parameter and the amount of change along a path is controlled by the DE parameters but other aspects of the `splatSimulatePaths` model are controlled by these parameters.

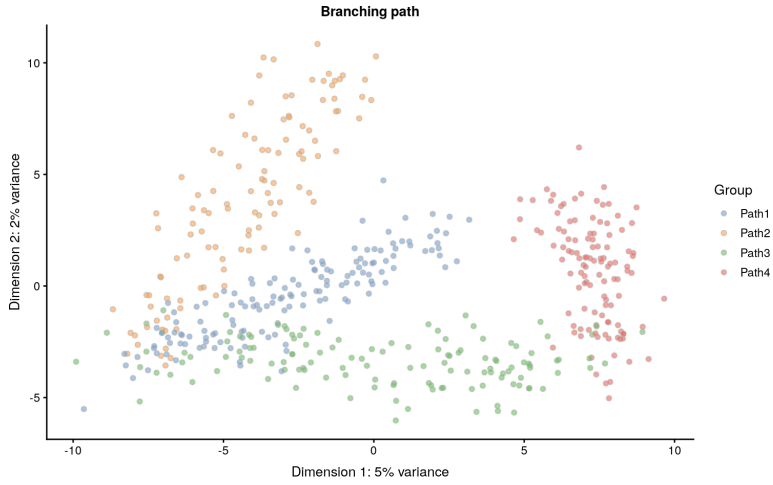
2.8.1 `path.from` - Path origin

This parameter controls the order of differentiation paths. It is a vector the same length as `group.prob` giving the starting position of each path. For example a `path.from` of `c(0, 1, 1, 3)` would indicate the Path 1 starts at the origin (0), Path 2 starts at the end of Path 1, Path 3 also starts at the end of Path 1 (a branch point) and Path 4 starts at the end of Path 3.

```
# Linear paths
sim1 <- splatSimulatePaths(params.groups,
                           group.prob = c(0.25, 0.25, 0.25,
0.25),
                           de.prob = 0.5, de.facLoc = 0.2,
                           path.from = c(0, 1, 2, 3),
                           verbose = FALSE)
sim1 <- normalize(sim1)
#> Warning in .local(object, ...): using library sizes as si
ze factors
plotPCA(sim1, colour_by = "Group") + ggtitle("Linear paths")
```



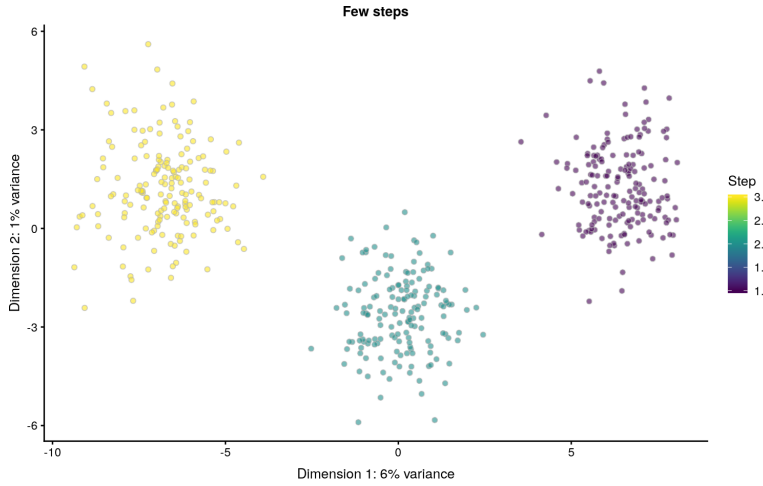

```
# Branching path
sim2 <- splatSimulatePaths(params.groups,
                           group.prob = c(0.25, 0.25, 0.25,
0.25),
                           de.prob = 0.5, de.facLoc = 0.2,
                           path.from = c(0, 1, 1, 3),
                           verbose = FALSE)
sim2 <- normalize(sim2)
#> Warning in .local(object, ...): using library sizes as si
ze factors
plotPCA(sim2, colour_by = "Group") + ggtitle("Branching pat
h")
```



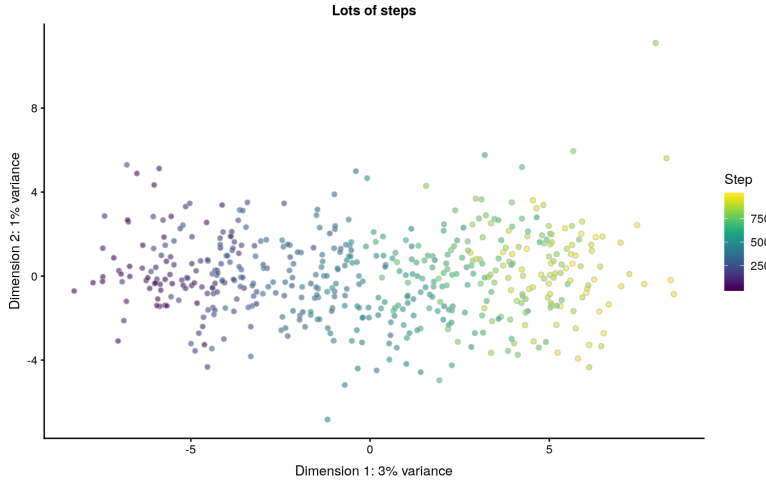
2.8.2 `path.nSteps` - Number of steps

A path is created by using the same differential expression procedure as used for groups to generate an end point. Interpolation is then used to create a series of steps between the start and end points. This parameter controls the number of steps along a path and therefore how discrete or smooth it is.

```
# Few steps
sim1 <- splatSimulatePaths(params.groups, path.nSteps = 3,
                           de.prob = 0.5, de.facLoc = 0.2, v
erbose = FALSE)
sim1 <- normalize(sim1)
#> Warning in .local(object, ...): using library sizes as si
ze factors
plotPCA(sim1, colour_by = "Step") + ggtitle("Few steps")
```



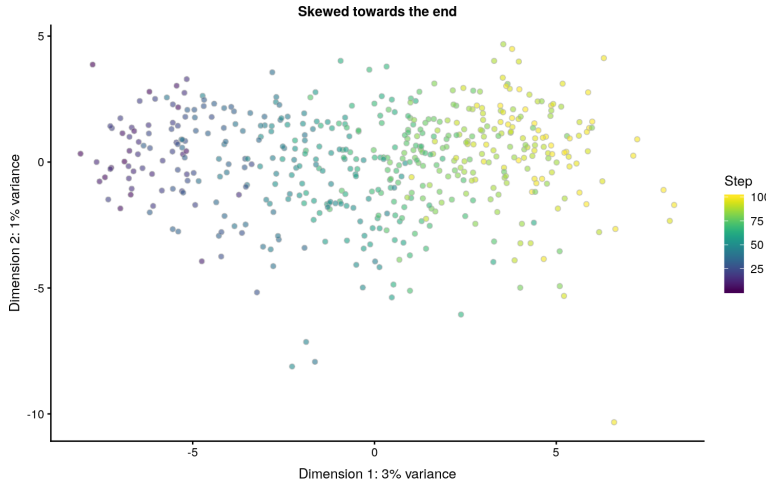

```
# Lots of steps
sim2 <- splatSimulatePaths(params.groups, path.nSteps = 1000
,
                           de.prob = 0.5, de.facLoc = 0.2, v
erbose = FALSE)
sim2 <- normalize(sim2)
#> Warning in .local(object, ...): using library sizes as si
ze factors
plotPCA(sim2, colour_by = "Step") + ggtitle("Lots of steps")
```



2.8.3 `path.skew` - Path skew

By default cells are evenly distributed along a path but it sometimes be useful to introduce a skew in the distribution, For example you may want to simulate a scenario with few stem-like cells and many differentiated cells. Setting `path.skew` to 0 will mean that all cells come from the end point while higher values up to 1 will skew them towards the start point.

```
# Skew towards the end
sim1 <- splatSimulatePaths(params.groups, path.skew = 0.1,
                           de.prob = 0.5, de.facLoc = 0.2, v
erbose = FALSE)
sim1 <- normalize(sim1)
#> Warning in .local(object, ...): using library sizes as si
ze factors
plotPCA(sim1, colour_by = "Step") + ggtitle("Skewed towards
 the end")
```



2.8.4 `path.nonlinearProb` - Non-linear probability

Most genes are interpolated in a linear way along a path but in reality this may not always be the case. For example it is easy to imagine a gene that is lowly-expressed at the start of a process, highly-expressed in the middle and lowly-expressed again at the end. The `path.nonlinearProb` parameter controls the probability that a gene will change in a non-linear way along a path.

2.8.5 `path.sigmaFac` - Path skew

Non-linear changes along a path are achieved by building a Brownian bridge between the two end points. A Brownian bridge is Brownian motion controlled in such a way that the end points are fixed. The `path.sigmaFac` parameter controls how extreme each step in the Brownian motion is and therefore how much the interpolation differs from a linear path.