SCRIP simulation for scRNA-seq data

Fei Qin

Last updated: 02/18/2021

1. Introduction to SCRIP method

SCRIP proposed two frameworks based on Gamma-Poisson and Beta-Poisson distribution for simulating scRNA-seq data. Both Gamma-Poisson and Beta-Poisson distribution model the over dispersion of scRNA-seq data. Specifically, Beta-Poisson model was used to model bursting effect. The dispersion was accurately simulated by fitting the mean-BCV dependency using generalized additive model (GAM). Other key characteristics of scRNA-seq data including library size, zero inflation and outliers were also modeled by SCRIP. With its flexible modeling, SCRIP enables various application for different experimental designs and goals including DE analysis, clustering analysis, trajectory-based analysis and bursting analysis

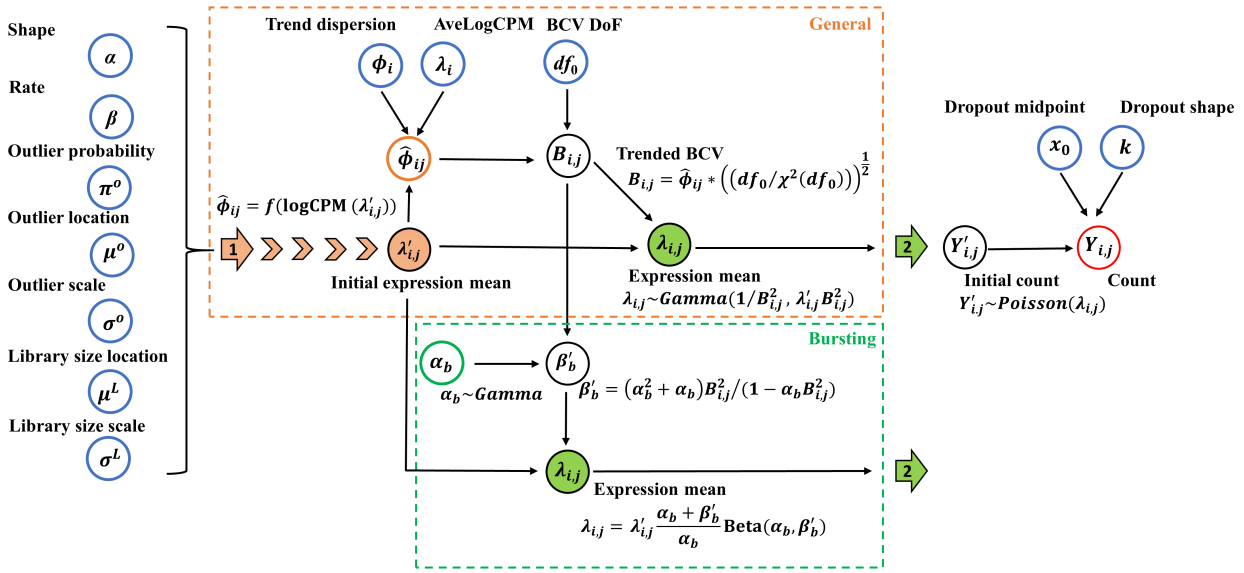


Figure 1: SCRIP framework

2. Installation

```
install.packages("TruncatedDistributions", repos="http://R-Forge.R-project.org")
BiocManager::install("splatter")

library(devtools)
install_github("FeiQin92/SCRIP")
```

3. Quick start

Assuming you already have a count matrix for scRNA-seq data, and you want to simulation data based on it. Only a few steps are needed to creat a simulation data using SCRIP.

A dataset from Xin data is used for example.

```
library(SCRIP)
library(Biobase)
library(splatter)

EMTAB.eset <- EMTABesethealthy
expre_data = exprs(EMTAB.eset)
pheno_data = pData(EMTAB.eset)

dim(expre_data)
```

```
## [1] 25453  1097
```

```
expre_data[1:6,1:6]
```

```
##         AZ_A10 AZ_A11 AZ_A12 AZ_A2 AZ_A5 AZ_A6
## SGIP1        0      0      0    32     0     0
## AZIN2        0      0      0     0     0     0
## CLIC4        3      0      0     1     0     0
## AGBL4        0      0      0     0     0     0
## NECAP2       0      0      0     0     0     0
## SLC45A1      0      0      0     0     0     0
```

```
head(pheno_data)
```

```
##        sampleID SubjectName cellTypeID cellType
## AZ_A10        1   Non T2D 1          5    delta
## AZ_A11        1   Non T2D 1          2    alpha
## AZ_A12        1   Non T2D 1          5    delta
## AZ_A2         1   Non T2D 1          9    gamma
## AZ_A5         1   Non T2D 1          6   ductal
## AZ_A6         1   Non T2D 1          2    alpha
```

```
# To simplify computation here, only acinar celltype in sample 5 was utilized here.
sub_expre_data=expre_data[,which((pheno_data$sampleID %in% c(5)) &
                                 (pheno_data[,4] == "acinar"))]
params <- splatEstimate(sub_expre_data)
```

```
sim_trend <-  SCRIPsimu(data=sub_expre_data, params=params, mode="GP-trendedBCV")
sim_trend
```

```
## class: SingleCellExperiment
## dim: 25453 80
## metadata(13): Params method ... batch.facScale bcv.shrink
## assays(5): BatchCellMeans BaseCellMeans CellMeans TrueCounts counts
## rownames(25453): Gene1 Gene2 ... Gene25452 Gene25453
## rowData names(4): Gene BaseGeneMean OutlierFactor GeneMean
## colnames(80): Cell1 Cell2 ... Cell79 Cell80
## colData names(3): Cell Batch ExpLibSize
## reducedDimNames(0):
## altExpNames(0):
```

4 Single cell type simulation

4.1 Parameter estimation

SCRIP utlized the estimation strategy from splatter, but also provided more parameters (Fold change, dropout rates, library size, BCV degree of freedom) to serve different experimental designs (i.e. Simulation for differential expression analysis, clustering analysis and trajectory analysis). Detailed description about other parameters will be shown in other sections of this document.

```
params <- splatEstimate(sub_expre_data)
params
```

```
## An object of class "SplatParams" (from package "splatter")
## Slot "nBatches":
## [1] 1
##
## Slot "batchCells":
## [1] 80
##
## Slot "batch.facLoc":
## [1] 0.1
##
## Slot "batch.facScale":
## [1] 0.1
##
## Slot "mean.shape":
## [1] 0.4702858
##
## Slot "mean.rate":
## [1] 0.05826439
##
## Slot "lib.loc":
## [1] 12.6806
##
## Slot "lib.scale":
## [1] 1.006336
##
## Slot "lib.norm":
## [1] FALSE
##
## Slot "out.prob":
## [1] 0.008469791
##
## Slot "out.facLoc":
## [1] 4.984882
##
## Slot "out.facScale":
## [1] 0.9876266
##
## Slot "nGroups":
## [1] 1
##
## Slot "group.prob":
```

```
## [1] 1
##
## Slot "de.prob":
## [1] 0.1
##
## Slot "de.downProb":
## [1] 0.5
##
## Slot "de.facLoc":
## [1] 0.1
##
## Slot "de.facScale":
## [1] 0.4
##
## Slot "bcv.common":
## [1] 0.4532238
##
## Slot "bcv.df":
## [1] 18.69833
##
## Slot "dropout.type":
## [1] "none"
##
## Slot "dropout.mid":
## [1] 0.8873834
##
## Slot "dropout.shape":
## [1] -1.148044
##
## Slot "path.from":
## [1] 0
##
## Slot "path.nSteps":
## [1] 100
##
## Slot "path.skew":
## [1] 0.5
##
## Slot "path.nonlinearProb":
## [1] 0.1
##
## Slot "path.sigmaFac":
## [1] 0.8
##
## Slot "nGenes":
## [1] 25453
##
## Slot "nCells":
## [1] 80
##
## Slot "seed":
## [1] 44945
##
## Batches:
```

```
##     [BATCHES]  [BATCH CELLS]     [Location]        [Scale]
##             1             80            0.1            0.1
##
## Mean:
##             (RATE)             (SHAPE)
## 0.0582643947909828   0.470285811951115
##
## Library size:
##       (LOCATION)           (SCALE)            (Norm)
## 12.6805964486986   1.0063358463285             FALSE
##
## Exprs outliers:
##       (PROBABILITY)           (LOCATION)              (SCALE)
## 0.00846979107848673     4.98488229571131    0.987626631086789
##
## 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)
## 0.453223761961301   18.6983255259522
##
## Dropout:
##            [Type]         (MIDPOINT)            (SHAPE)
##              none  0.887383387745456  -1.14804417937622
##
## Paths:
##         [From]         [Steps]          [Skew]    [Non-linear]  [Sigma Factor]
##              0             100             0.5             0.1             0.8
```

4.2 Simulation

The default mode in SCRIP for simulation is "GP-trendedBCV". You can also choose other modes ("GP-commonBCV", "BP-commonBCV", "BP-trendedBCV", "BP-burstBCV" and "BP") in the SCRIPsimu() function. Specifically, for the "BP" mode, you have to provide the "bursting" information (burst frequency and burst size), which was estimated from directly from real data by performing a profile-likelihood method built up on Beta-Poisson method proposed by Larsson et al.(not shown here). For single cell type simulation, you have to set the "method" as "single", which was default in SCRIPsimu() function.

4.2.1 GP-commonBCV

GP-commonBCV is the model used by splatter. GP-commonBCV applied the Gamma-Poisson mixture model with mean-BCV dependency fitted by a common BCV across genes.

```
########################## GP-commonBCV model/Splatter ##########################
#################################################################################
sim_GPcommon <-  SCRIPsimu(data=sub_expre_data, params=params, mode="GP-commonBCV")
sim_GPcommon
```

```
## class: SingleCellExperiment
## dim: 25453 80
## metadata(13): Params method ... batch.facScale bcv.shrink
## assays(5): BatchCellMeans BaseCellMeans CellMeans TrueCounts counts
## rownames(25453): Gene1 Gene2 ... Gene25452 Gene25453
## rowData names(4): Gene BaseGeneMean OutlierFactor GeneMean
## colnames(80): Cell1 Cell2 ... Cell79 Cell80
## colData names(3): Cell Batch ExpLibSize
## reducedDimNames(0):
## altExpNames(0):
```

4.2.2 GP-trendedBCV

GP-trendedBCV is the major model of SCIRP. which used the Gamma-Poisson mixture model with mean-BCV dependency fitted by GAM.

```
############################## GP-trendedBCV model ##############################
#################################################################################
sim_GPtrend <-  SCRIPsimu(data=sub_expre_data, params=params, mode="GP-trendedBCV")
```

4.2.3 BP-commonBCV

BP-commonBCV is the model used for simulating bursting effect with Beta-Poisson mixture distribution. The mean-BCV dependency was fitted by a common BCV across genes.

```
############################## BP-commonBCV model ###############################
#################################################################################
sim_BPcommon <-  SCRIPsimu(data=sub_expre_data, params=params, mode="BP-commonBCV")
```

4.2.4 BP-trendedBCV

BP-trendedBCV is the model used for simulating bursting effect with Beta-Poisson mixture distribution. The mean-BCV dependency was fitted by a GAM.

```
############################## BP-trendedBCV model ##############################
#################################################################################
sim_BPtrend <-  SCRIPsimu(data=sub_expre_data, params=params, mode="BP-trendedBCV")
```

4.2.5 BP-burstBCV

BP-burstBCV is also the model used for simulating bursting effect with Beta-Poisson mixture distribution. The mean-BCV dependency was fitted by a GAM. In "BP-trendedBCV" from 4.2.4, we assumed that bursting effect was nested in BCV which has the dependency with mean. However for "BP-burstBCV" here, we assumed that mean-BCV dependency and bursting effect were independent. Under this assumption, we simulated underlying mean count from beta distribution and included BCV information in the Beta-Negative Binomial framework using the GAM function described above.

```
############################## BP-trendedBCV model ##############################
#################################################################################
sim_BPtrend_burst <-  SCRIPsimu(data=sub_expre_data, params=params, mode="BP-burstBCV")
```

5 Group simulation

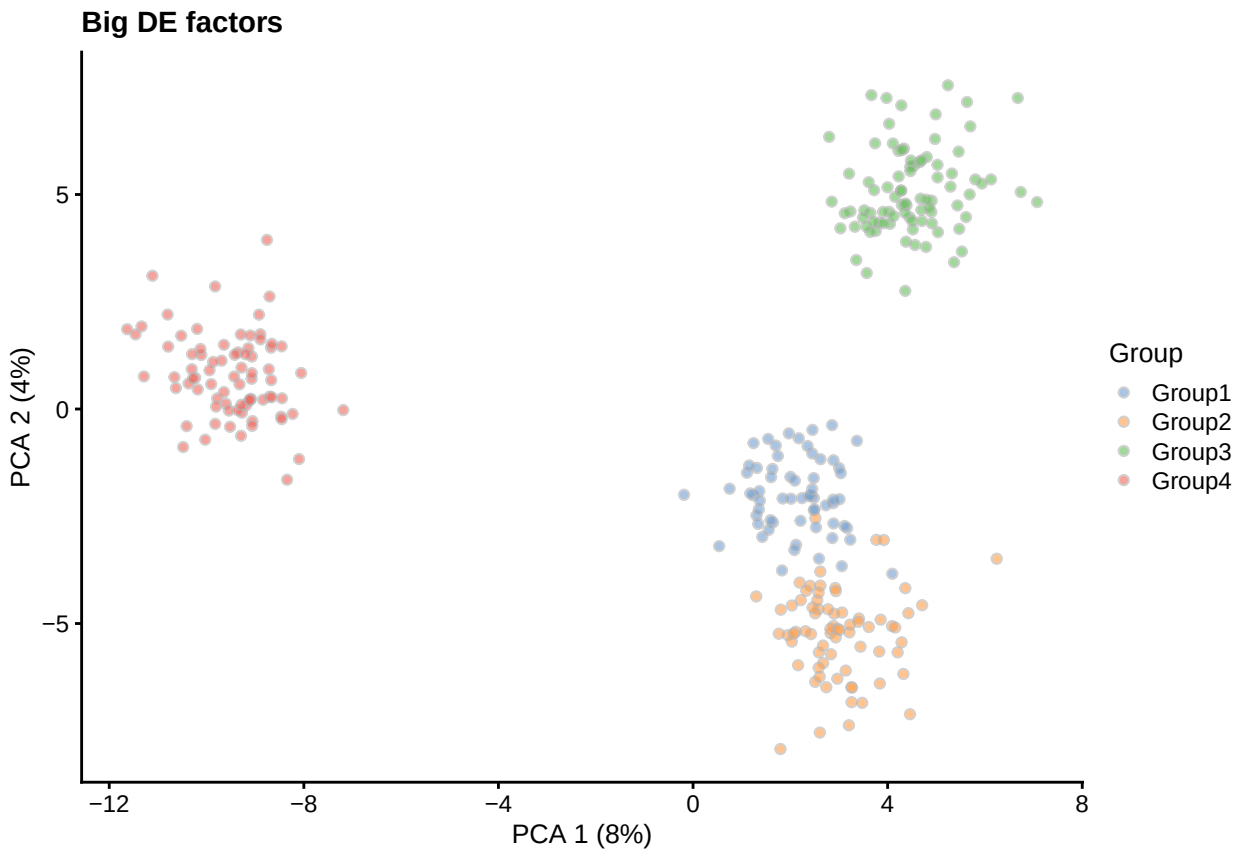
Group simulation is useful for studying different experimental conditions, especially for differential expression (DE) analysis. To serve different applications in scRNA-seq analysis, SCRIP provides flexible simulation. It can simulate scRNA-seq data with different parameters from multiple cell groups (i.e. cell types), which is useful for evaluating the detection of global characteristics such as clustering. It also allows simulation of group difference in a single cell group, which is useful for evaluating typical DE analysis methods.

5.1 Basic group simulation

DEGs were simulated using multiplicative differential expression factors from a log-normal distribution with parameters including number of genes (nGenes), the path-specific proportion of DE genes (de.prob), the proportion of down-regulated DE genes (de.downProb), DE location factor (de.facLoc) and DE scale factor (de.facScale).

```
counts <- as.matrix(Tung)

counts <- counts[1:10000,1:200]
params <- splatEstimate(counts)
```

```
sim.SCRIP2 <- SCRIPsimu(data=counts, params=params, method="groups",
                        batchCells=300, group.prob = c(0.25, 0.25, 0.25, 0.25),
                        de.prob = c(0.2, 0.2, 0.2, 0.2),
                        de.downProb = c(0.5, 0.5, 0.5, 0.5),
                        de.facLoc = c(0.2, 0.3, 0.4, 0.5),
                        de.facScale=c(0.2, 0.2, 0.2, 0.2))
library(scater)
sim2 <- logNormCounts(sim.SCRIP2)
sim2 <- runPCA(sim2)
plotPCA(sim2, colour_by = "Group") + ggtitle("Big DE factors")
```

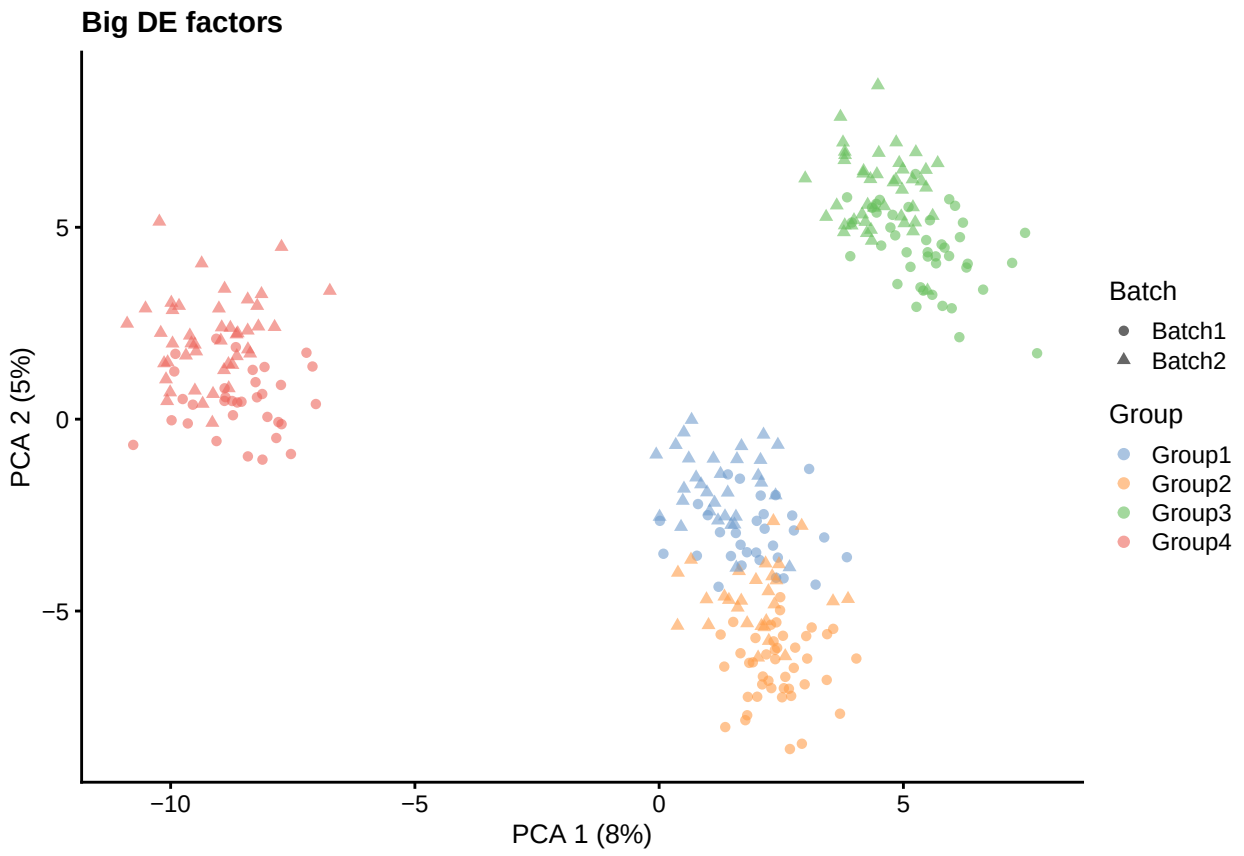


5.2 Group simulation with batch effect

Batch effect factors are also generated from a log-normal distribution with parameters including batchCells, batch.facLoc and batch.facScale.

batchCells: number of cells for each batch
batch.facLoc: Batch location factor in log-normal distribution for batch factor
batch.facScale: Batch scale factor in log-normal distribution for batch factor

```
sim.SCRIP3 <- SCRIPsimu(data=counts, params=params, method="groups",
                        batchCells=c(150, 150),
                        batch.facLoc = c(0.1, 0.1),
                        batch.facScale = c(0.1, 0.1),
                        group.prob = c(0.25, 0.25, 0.25, 0.25),
                        de.prob = c(0.2, 0.2, 0.2, 0.2),
                        de.downProb = c(0.5, 0.5, 0.5, 0.5),
                        de.facLoc = c(0.2, 0.3, 0.4, 0.5),
                        de.facScale=c(0.2, 0.2, 0.2, 0.2))
sim3 <- logNormCounts(sim.SCRIP3)
sim3 <- runPCA(sim3)
plotPCA(sim3, colour_by = "Group", shape_by="Batch") + ggtitle("Big DE factors")
```



5.3 Group simulation for clustering

To simulate multi-cell type scRNA-seq data, we estimated hyperparameters from real data and simulate data for each cell type separately. SCRIP preserved cluster discriminative genes, which is not capable in available methods. We identified variably expressed genes (VEGs) from each cell type in real data. For these VEGs, true cell mean was set to their expression means in real data to simulate cell-type discriminative genes. Whereafter, aforementioned simulation steps (simulating true cell mean; simulating true counts) will be followed to complete the simulation.

```
### Xin ####
EMTAB.eset <- EMTABesethealthy
library(Biobase)
library(edgeR)
library(stats)
expre_data = exprs(EMTAB.eset)
pheno_data=pData(EMTAB.eset)
expre_data <- as.matrix(expre_data[1:2000,])

expre_data[1:6,1:6]
```

```
##         AZ_A10 AZ_A11 AZ_A12 AZ_A2 AZ_A5 AZ_A6
## SGIP1        0      0      0    32     0     0
## AZIN2        0      0      0     0     0     0
## CLIC4        3      0      0     1     0     0
## AGBL4        0      0      0     0     0     0
```

```
## NECAP2       0      0      0     0     0     0
## SLC45A1      0      0      0     0     0     0
```

```
head(pheno_data)
```

```
##        sampleID SubjectName cellTypeID cellType
## AZ_A10        1   Non T2D 1          5    delta
## AZ_A11        1   Non T2D 1          2    alpha
## AZ_A12        1   Non T2D 1          5    delta
## AZ_A2         1   Non T2D 1          9    gamma
## AZ_A5         1   Non T2D 1          6   ductal
## AZ_A6         1   Non T2D 1          2    alpha
```

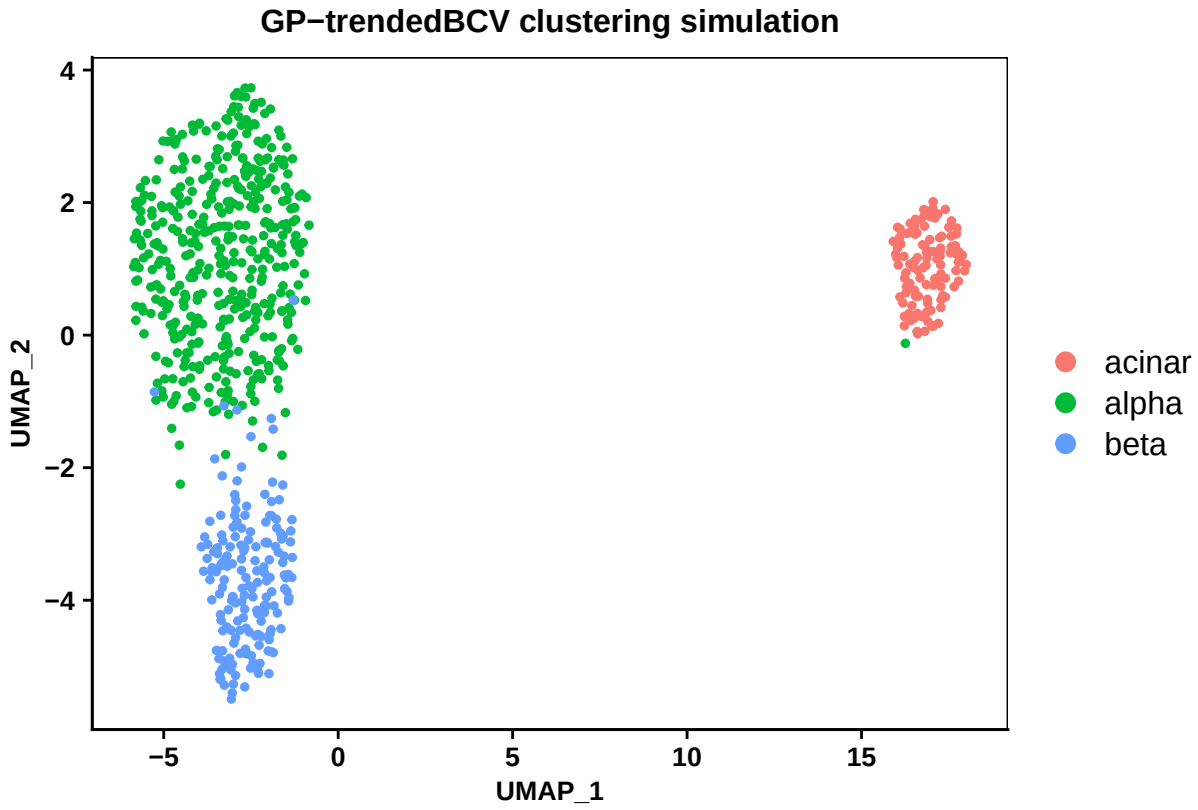
```
table(pheno_data$cellType)
```

```
##
##                 acinar                  alpha                   beta
##                    112                    443                    171
##          co-expression                  delta                 ductal
##                     26                     59                    135
##            endothelial                epsilon                  gamma
##                     13                      5                     75
##                   mast           MHC class II                    PSC
##                      4                      1                     23
##           unclassified unclassified endocrine
##                      1                     29
```

```
CTlist <- c("acinar","alpha","beta")

params <- splatEstimate(expre_data)
```

```
library(Seurat)
final.list <- simu_cluster(expre_data = expre_data, pheno_data = pheno_data, CT=CTlist,
                           mode="GP-trendedBCV", nfeatures=100)

final <- final.list$final
CT.infor <- final.list$CT.infor

library(ggplot2)
Group.sim <- function(data){
  library(dplyr)
  library(Seurat)
  library(cowplot)
  pbmc <- CreateSeuratObject(counts = data, project = "MuSiC", min.cells = 3,
                             min.features = 200)
  pbmc <- NormalizeData(pbmc, normalization.method = "LogNormalize", scale.factor = 10000)
  pbmc <- FindVariableFeatures(pbmc, selection.method = "vst", nfeatures = 100)
  all.genes <- rownames(pbmc)
  pbmc <- ScaleData(pbmc, features = all.genes)
  pbmc <- RunPCA(pbmc, features = VariableFeatures(object = pbmc))
  pbmc <- RunUMAP(pbmc, dims = 1:10)
  pbmc <- RunTSNE(pbmc)
```

```
  return(pbmc)
}

pbmc <- Group.sim(data=final)
pbmc@meta.data$Celltype <- CT.infor
umap <- DimPlot(pbmc, reduction = "umap",group.by = "Celltype",pt.size=1)+
  labs(title="GP-trendedBCV clustering simulation")+
  theme(plot.title = element_text(face="bold",size=12, hjust=0.5),
        legend.text=element_text(size=12),
        axis.text=element_text(size=10,face="bold"),
        axis.title.x = element_text(size=10,face="bold"),
        axis.title.y = element_text(size=10,face="bold"),
        panel.background = element_rect(fill='white',colour="black"))

umap
```



5.4 Group simulation for DE analysis

To simulate single-cell type scRNA-seq data, SCRIP enabled to simulate with different DE rates, fold change, dropout rate, library size, BCV.df and BCV.shrink. You change these parameters in simu_DE() function, which will finally export the simulation results for group1 and group2, as well as the true DEGs.

```
## Here I will use Tung dataset (given in package) to give examples about how to change
## parameter for evaluating DE analysis methods.
```

```
counts <- as.matrix(Tung)
library(Biobase)
library(gtools)
library(lcmix)
## To simplify computation, only 1000 genes and 100 cells from Tung dataset will be use
## for estimation and simulation
expre_data_sub <- counts[1:1000,1:100]
params1 <- splatEstimate(expre_data_sub)
```

1) Fold change: FC enables us to control the difference between two groups. Larger fold change value means larger pre-defined difference between two groups; it is expected that we can find better performance for each DE analysis tool.

```
################### Change the parameter of fold change (FC) ####################
#################################################################################
resFC <- simu_DE(expre_data = expre_data_sub, params = params1, nDE=50, FC=1.4)
## simu_DE() function will export the simulation results for group1 and group2,
## as well as the true DEGs.
resFC
```

```
## [[1]]
## class: SingleCellExperiment
## dim: 1000 100
## metadata(14): Params method ... batch.facScale bcv.shrink
## assays(5): BatchCellMeans BaseCellMeans CellMeans TrueCounts counts
## rownames(1000): Gene1 Gene2 ... Gene999 Gene1000
## rowData names(4): Gene BaseGeneMean OutlierFactor GeneMean
## colnames(100): Cell1 Cell2 ... Cell99 Cell100
## colData names(3): Cell Batch ExpLibSize
## reducedDimNames(0):
## altExpNames(0):
##
## [[2]]
## class: SingleCellExperiment
## dim: 1000 100
## metadata(14): Params method ... batch.facScale bcv.shrink
## assays(5): BatchCellMeans BaseCellMeans CellMeans TrueCounts counts
## rownames(1000): Gene1 Gene2 ... Gene999 Gene1000
## rowData names(4): Gene BaseGeneMean OutlierFactor GeneMean
## colnames(100): Cell1 Cell2 ... Cell99 Cell100
## colData names(3): Cell Batch ExpLibSize
## reducedDimNames(0):
## altExpNames(0):
##
## [[3]]
##  [1] "gene341" "gene102" "gene316" "gene314" "gene206" "gene630" "gene474"
##  [8] "gene908" "gene337" "gene931" "gene202" "gene813" "gene441" "gene446"
## [15] "gene653" "gene947" "gene470" "gene533" "gene900" "gene522" "gene338"
## [22] "gene780" "gene806" "gene485" "gene4"   "gene526" "gene428" "gene348"
## [29] "gene64"  "gene128" "gene223" "gene773" "gene907" "gene677" "gene687"
## [36] "gene963" "gene895" "gene216" "gene878" "gene471" "gene319" "gene21"
## [43] "gene449" "gene705" "gene569" "gene131" "gene798" "gene537" "gene703"
## [50] "gene204"
```

2) Dropout rate: dropout rate controls the zero proportion in scRNA-seq data, since zero inflation is one of the main characteristics in scRNA-seq data.

```
########### ##### Change the parameter of dropout rates (Dropout_rate) ##########
###############################################################################
resDrop <- simu_DE(expre_data = expre_data_sub, params = params1, nDE=50, FC=1.4,
                   Dropout_rate=0.2)
```

3) Library size: library size can be used to control the expression level for each gene, as well as the zero proportions in the scRNA-seq data.

```
################ Change the parameter of library size (libsize) #################
###############################################################################
resLib <- simu_DE(expre_data = expre_data_sub, params = params1, nDE=50, FC=1.4,
                  libsize=5000)
```

4) BCV.df: BCV.df enables us to change the variation of BCV values. Smaller variation is given to BCV values when BCV.df gets larger.

```
########## Change the parameter of BCV degree of freedom (pre.bcv.df) ###########
###############################################################################
resDf <- simu_DE(expre_data = expre_data_sub, params = params1, nDE=50, FC=1.4,
                 pre.bcv.df=5)
```

5) BCV.shrink: This parameter is introduced to amplify (> 1) or shrink (< 1) BCV values, which mean it can control the BCV levels.

```
######### Change the parameter of BCV shrinkage factor (bcv.shrink) #############
###############################################################################
resShrink <- simu_DE(expre_data = expre_data_sub, params = params1, nDE=50, FC=1.4,
                     bcv.shrink = 1.5)
```

Here I take fold change as an example to introduce how to use the simulation data to evaluate DE analysis methods (i.e. edgeR, DESeq2, limma voom and MAST):

```
sim=resFC[[1]]
simDE=resFC[[2]]
genenameDE=resFC[[3]]

exps <- counts(sim)
expsDE <- counts(simDE)
counts <- cbind(exps,expsDE)

colnames(counts) <- paste0("cell",1:ncol(counts))
rownames(counts) <- paste0("gene",1:nrow(counts))

truth <- rownames(counts)
truth[which(rownames(counts) %in% genenameDE)] <- "DE"
truth[-which(rownames(counts) %in% genenameDE)] <- "common"
truth <- as.factor(truth)
```

As an application of SCRIP, we simulated scRNA-seq data using SCRIP and compared four DE analysis tools including edgeR, DESeq2, Limma-voom and MAST. To better account for zero inflation of scRNA-seq data in DE analysis, a weighting strategy of ZINB-WaVE was also integrated with edgeR, DESeq2 and Limma-voom.

ZINB WAVE: ZINB-WaVE is used to computes gene- and cell-specific weights, based on zero-inflated negative binomial model, to "unlock" bulk RNA-seq tools for single-cell applications.

```
##############################################################################
################################## ZINB WAVE #################################
library(zinbwave)
library(DESeq2)
group <- factor(c(rep(1,ncol(exps)),rep(2,ncol(expsDE))))
coldata <- data.frame(condition=group)
rownames(coldata) <- colnames(counts)
fluidigm <- DESeqDataSetFromMatrix(countData = counts,
                                   colData = coldata,
                                   design = ~ condition)
zinb <- zinbFit(fluidigm, K=2, epsilon=1000)
fluidigm_zinb <- zinbwave(fluidigm, fitted_model = zinb, K = 2, epsilon=1000,
                          observationalWeights = TRUE)
ZINB.Wave.weight <- assay(fluidigm_zinb, "weights")
```

```
ZINB.Wave.weight[1:6,1:6]
```

```
##       cell1       cell2     cell3 cell4 cell5     cell6
## gene1     1 0.032917511 1.0000000     1     1 0.8630931
## gene2     1 0.005597185 1.0000000     1     1 1.0000000
## gene3     1 0.022043928 0.8493267     1     1 1.0000000
## gene4     1 0.175001116 1.0000000     1     1 1.0000000
## gene5     1 1.000000000 1.0000000     1     1 1.0000000
## gene6     1 1.000000000 1.0000000     1     1 1.0000000
```

5.4.1 edgeR

edgeR was originally designed for bulk RNA sequencing data and it used a overdispersed Poisson model and an Empirical Bayes modelto accout for both biological and technical variability.

```
##############################################################################
################################### edgeR ####################################
library(edgeR)
dgList <- DGEList(assay(fluidigm))
countsPerMillion <- cpm(dgList)
countCheck <- countsPerMillion > 1
keep <- which(rowSums(countCheck) >= 2)
dgList <- dgList[keep,]
dgList <- calcNormFactors(dgList, method="TMM")
design <- model.matrix(~condition, data = colData(fluidigm))
dgList <- estimateDisp(dgList, design)
fit <- glmFit(dgList, design)
lrt <- glmLRT(fit, coef=2)
edgeR.res <- as.data.frame(topTags(lrt,n=nrow(lrt$coefficients)))
edgeR.res <- edgeR.res[rownames(counts),]
```

```
head(edgeR.res)
```

```
##             logFC    logCPM          LR       PValue          FDR
## gene1  0.13177586  9.825929  0.65865936 4.170336e-01 0.9999901758
## gene2 -0.08197831 10.479397  0.56271182 4.531697e-01 0.9999901758
## gene3 -0.01356489 10.336600  0.01359588 9.071760e-01 0.9999901758
## gene4  0.53449219 10.275283 20.10589822 7.327037e-06 0.0005763936
## gene5 -0.01093406 11.026393  0.01494481 9.027019e-01 0.9999901758
## gene6 -0.01800622 11.726386  0.06722908 7.954148e-01 0.9999901758
```

5.4.2 edgeR ZINB Wave

To apply edgeR for scRAN-seq data, we computed gene- and cell-specific weights using ZINB Wave approach. Then such weights were considered in the DE analysis.

```
################################################################################
################################ edgeR ZINB WAVE ###############################

library(edgeR)
dge <- DGEList(assay(fluidigm_zinb))
dge <- calcNormFactors(dge)

design <- model.matrix(~condition, data = colData(fluidigm))
dge$weights <- weights
dge <- estimateDisp(dge, design)
fit <- glmFit(dge, design)
lrt <- glmWeightedF(fit, coef = 2)
edgeR.weighted.res <- as.data.frame(topTags(lrt,n=nrow(lrt$coefficients)))
edgeR.weighted.res <- edgeR.weighted.res[rownames(counts),]
```

```
head(edgeR.weighted.res)
```

```
##              logFC   logCPM           LR       PValue   padjFilter          FDR
## gene1  0.142326043  9.86776 8.234834e-01 3.641667e-01 0.9790832329 0.9938617899
## gene2 -0.075053733 10.49116 4.876433e-01 4.849827e-01 0.9790832329 0.9938617899
## gene3  0.002504339 10.35744 4.578038e-04 9.829295e-01 0.9913673479 0.9938617899
## gene4  0.537865543 10.28877 2.088248e+01 4.891543e-06 0.0002712583 0.0004291218
## gene5 -0.002699928 11.02853 9.533279e-04 9.753685e-01 0.9883302169 0.9938617899
## gene6 -0.010197471 11.72794 2.191736e-02 8.823076e-01 0.9790832329 0.9938617899
```

5.4.3 DESeq2

DESeq2, originally designed for bulk RNA-seq data, used a shrinkage estimation approach for dispersions and fold change to improve the stability and interpretability of estimates in DE analysis

```
################################################################################
################################## DESeq2 ######################################
library(DESeq2)
dds <- DESeqDataSet(fluidigm, design = ~ condition)
dds <- DESeq(dds, sfType="poscounts", useT=TRUE, minmu=1e-6)
```

```
DESeq2.res <- results(dds)
DESeq2.res <- as.data.frame(DESeq2.res)
DESeq2.res <- DESeq2.res[rownames(counts),]
```

```
head(DESeq2.res)
```

```
##        baseMean log2FoldChange      lfcSE        stat       pvalue        padj
## gene1  1.465990    0.145570325 0.17756700  0.81980505 4.133130e-01 0.993320048
## gene2  3.402359   -0.080559897 0.11621461 -0.69319937 4.889968e-01 0.993320048
## gene3  2.852846   -0.008349764 0.12594954 -0.06629451 9.472102e-01 0.993320048
## gene4  2.708543    0.562899626 0.13126374  4.28831019 2.811717e-05 0.002129875
## gene5  5.829782   -0.007688601 0.09255534 -0.08307031 9.338796e-01 0.993320048
## gene6 10.758111   -0.014440885 0.06931949 -0.20832360 8.351904e-01 0.993320048
```

5.4.4 DESeq2 ZINB Wave

```
##############################################################################
############################## DESeq2 ZINB WAVE ##############################
library(DESeq2)
dds <- DESeqDataSet(fluidigm_zinb, design = ~ condition)
dds <- DESeq(dds, sfType="poscounts", useT=TRUE, minmu=1e-6)

DESeq2.weighted.res <- results(dds)
DESeq2.weighted.res <- as.data.frame(DESeq2.weighted.res)
DESeq2.weighted.res <- DESeq2.weighted.res[rownames(counts),]
```

```
head(DESeq2.weighted.res)
```

```
##        baseMean log2FoldChange      lfcSE        stat       pvalue         padj
## gene1  1.465990    0.149375861 0.17230793  0.86691229 3.871163e-01           NA
## gene2  3.402359   -0.081861071 0.11399693 -0.71809888 4.735546e-01 0.9876024946
## gene3  2.852846    0.001202566 0.12294403  0.00978141 9.922058e-01 0.9945644631
## gene4  2.708543    0.557659981 0.12886718  4.32740123 2.410319e-05 0.0009685101
## gene5  5.829782   -0.007977674 0.09180753 -0.08689564 9.308423e-01 0.9876024946
## gene6 10.758111   -0.014460525 0.06922241 -0.20889947 8.347415e-01 0.9876024946
```

5.4.5 limma voom

Voom model estimated the mean-variance relationship for the log-counts, which then generated a precision weight for each observation. Then these weights were considered into the limma empirical Bayes analysis pipeline.

```
################################################  ##############################
################################### limma voom ###################################
library(edgeR)
d0 <- DGEList(counts)
d0 <- calcNormFactors(d0)

mm <- model.matrix(~0 + group)
```

```
y <- voom(d0, mm, plot = F)

fit <- lmFit(y, mm)

contr <- makeContrasts(group1 - group2, levels = colnames(coef(fit)))
tmp <- contrasts.fit(fit, contr)
tmp <- eBayes(tmp)
top.table <- topTable(tmp, sort.by = "none", n = Inf)
limma.res <- top.table
limma.res <- limma.res[rownames(counts),]
```

```
head(limma.res)
```

```
##              logFC   AveExpr           t      P.Value   adj.P.Val         B
## gene1 -0.116885509  8.711641 -0.80190655 4.234405e-01 0.994917404 -5.664062
## gene2  0.018612508  9.817581  0.15495400 8.769944e-01 0.994917404 -6.110713
## gene3  0.001664018  9.563575  0.01319705 9.894821e-01 0.997719862 -6.066216
## gene4 -0.535625452  9.478810 -4.11118509 5.491736e-05 0.004080783  1.676779
## gene5  0.025481405 10.589409  0.28366737 7.769222e-01 0.994917404 -6.292431
## gene6  0.002990824 11.482066  0.04362762 9.652393e-01 0.997719862 -6.605788
```

5.4.6 limma voom ZINB Wave

```
################################################################################
################################ limma voom ZINB WAVE ##########################
library(edgeR)
d0 <- DGEList(counts)
d0 <- calcNormFactors(d0)

mm <- model.matrix(~0 + group)
y <- voom(d0, mm, plot = F)
y$weights <- weights*y$weights

fit <- lmFit(y, mm)

contr <- makeContrasts(group1 - group2, levels = colnames(coef(fit)))
tmp <- contrasts.fit(fit, contr)
tmp <- eBayes(tmp)
top.table <- topTable(tmp, sort.by = "none", n = Inf)
limma.weighted.res <- top.table
limma.weighted.res <- limma.weighted.res[rownames(counts),]
```

```
head(limma.weighted.res)
```

```
##              logFC   AveExpr           t      P.Value   adj.P.Val         B
## gene1 -0.127859008  8.711641 -0.91834849 3.594914e-01 0.995248526 -6.191824
## gene2  0.022182187  9.817581  0.19246617 8.475633e-01 0.995248526 -6.780304
## gene3 -0.015433230  9.563575 -0.13060773 8.962108e-01 0.995248526 -6.726034
## gene4 -0.531702607  9.478810 -4.20996372 3.788892e-05 0.002641178  1.697347
## gene5  0.025481405 10.589409  0.28625093 7.749685e-01 0.995248526 -6.979007
## gene6  0.002990824 11.482066  0.04397005 9.649701e-01 0.995248526 -7.299061
```

5.4.7 MAST

MAST was designed specifically for scRNA-seq data. It developed a two-part, generalized linear model to parameterizes the characteristics in scRNA-seq data.

```
##############################################################################
################################### MAST #####################################

group <- factor(c(rep("Group0",ncol(exps)),rep("Group1",ncol(expsDE))))
group <- relevel(group,"Group0")
coldata <- data.frame(condition=group)
rownames(coldata) <- colnames(counts)
fluidigm <- DESeqDataSetFromMatrix(countData = counts,
                                   colData = coldata,
                                   design = ~ condition)
library(MAST)
library(data.table)
sca <- FromMatrix(counts,data.frame(condition=group),
                  data.frame(Geneid=rownames(counts)),check_sanity = FALSE)
zlmCond <- zlm(~condition, sca)

summaryCond <- summary(zlmCond, doLRT='conditionGroup1')
print(summaryCond, n=4)
summaryDt <- summaryCond$datatable
fcHurdle <- merge(summaryDt[contrast=='conditionGroup1' & component=='H',.(primerid, `Pr(>Chisq)`)],
                  #hurdle P values
                  summaryDt[contrast=='conditionGroup1' & component=='logFC',
                            .(primerid, coef, ci.hi, ci.lo)],
                  by='primerid') #logFC coefficients
MAST.res=fcHurdle
MAST.res <- MAST.res[rownames(counts),]
```

```
head(MAST.res)
```

```
##    primerid   Pr(>Chisq)        coef     ci.hi      ci.lo
## 1:    gene1 0.6922000231  0.14826490 0.5569901 -0.2604603
## 2:    gene2 0.3852710888 -0.21669278 0.4519854 -0.8853710
## 3:    gene3 0.9711400690 -0.03878229 0.5765938 -0.6541583
## 4:    gene4 0.0006622683  1.08458107 1.6412395  0.5279226
## 5:    gene5 0.9884721335 -0.07996863 0.9520346 -1.1119718
## 6:    gene6 0.9734185035 -0.17992941 1.3433665 -1.7032253
```

For each method, true positive rates and false positive rates of DE detections can be calculated based on the knowledge of true and false DEGs. Receiver operating characteristic (ROC) curve and the areas under the ROC curve (AUC) can be used as measures to assess the performance of these DE analysis methods.

6 Cell lineage simulation

RNA transcriptomes data can be used to trace cell divisions and migrations over time, which is also referred to as cell lineage analysis. The developmental order of cells can be represented by a quantitative measure, pseudotime. To simulate pseudotime effect, we used the strategy of Splat based on differential expression process in cell developmental paths.Similarly to group simulation, in such differential expression process, we first defined the start and end expression level for each path. DEGs were simulated using multiplicative differential expression factors from a log-normal distribution with parameters including number of genes (nGenes), the path-specific proportion of DE genes (de.prob), the proportion of down-regulated DE genes (de.downProb), DE location factor (de.facLoc) and DE scale factor (de.facScale). Specifically, a few other parameters are provided to control the pattern of the lineages.

1) path.nSteps
   – number of steps between the start point and end point for each path

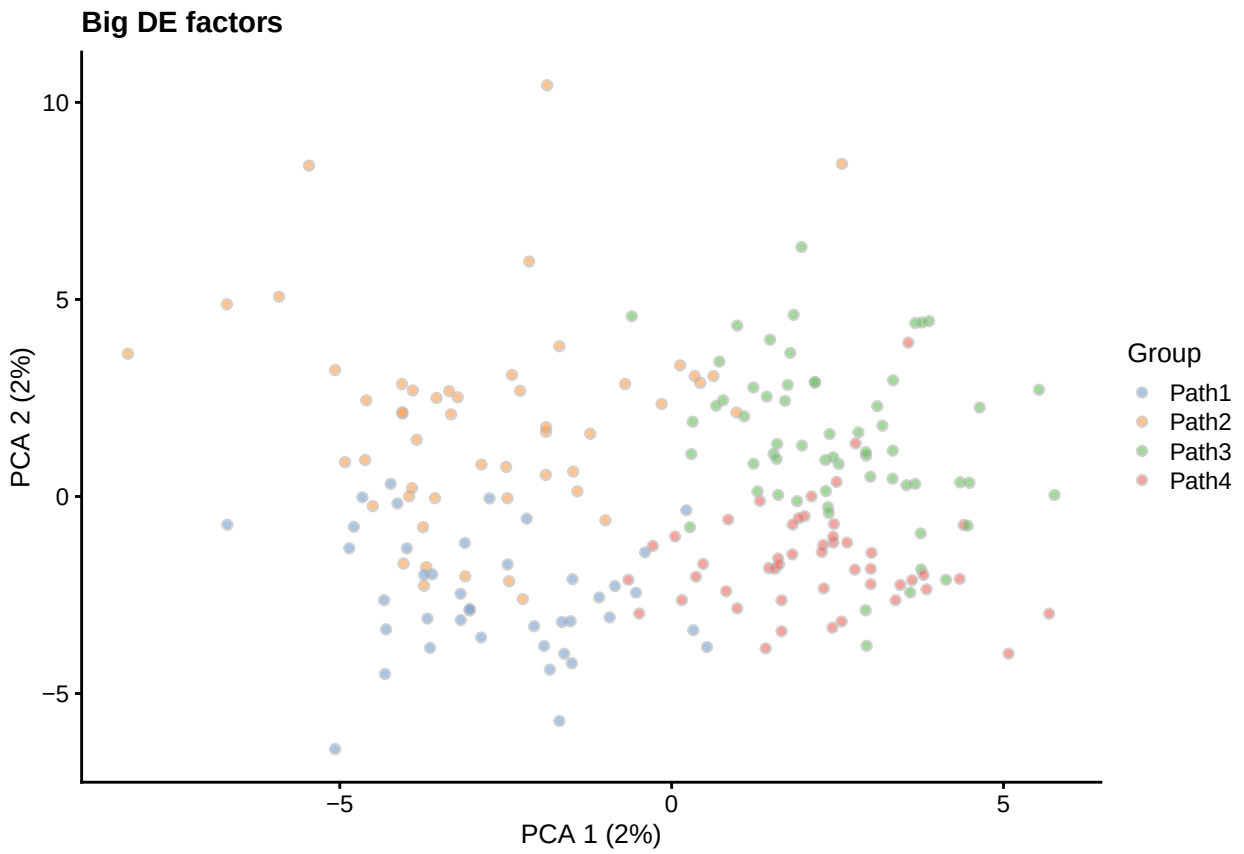
2) path.skew
   – Controls how likely cells are from the start or end point of the path. Cells are more likely to come from end point when path.skew gets close to 0, while cells are more likely to come from start point when path.skew gets close to 1.

3) path.from
   – Control the relationship between these paths.

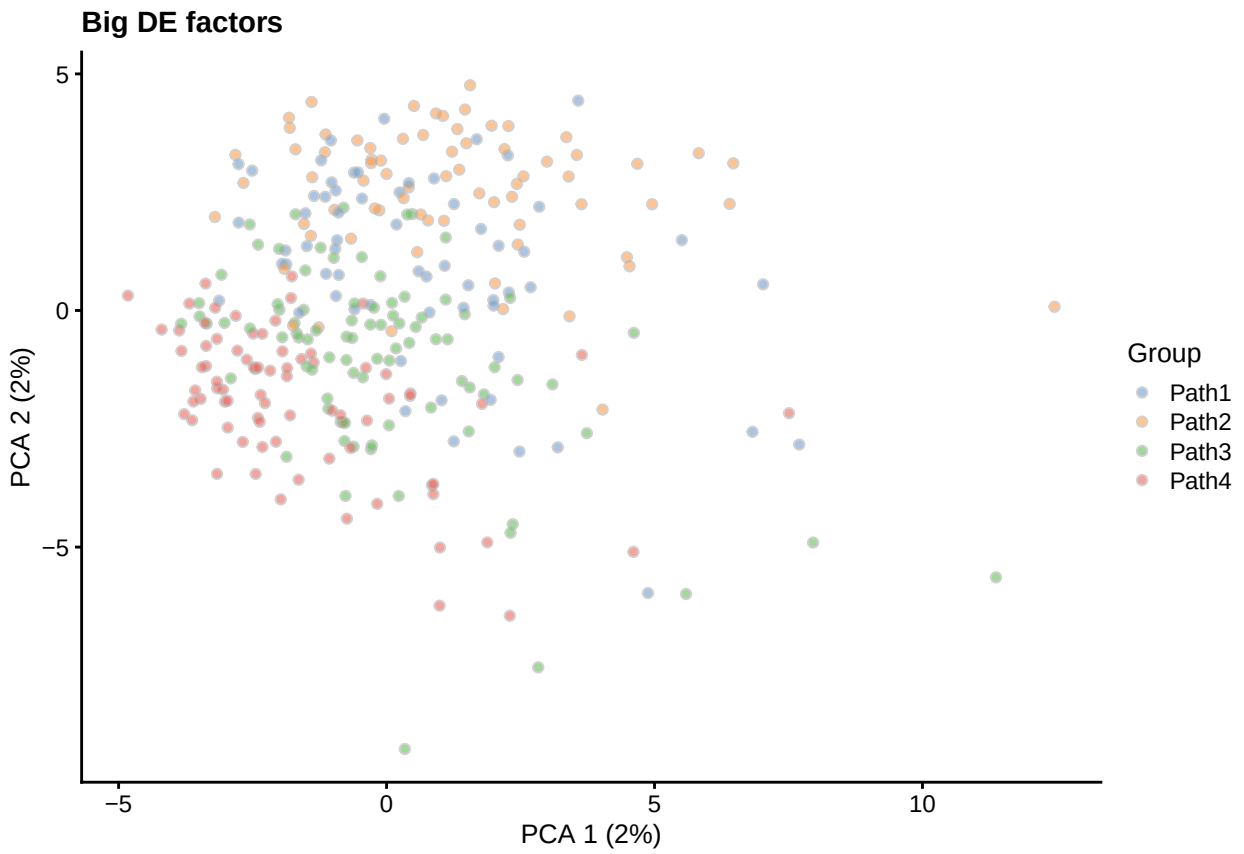
```
counts <- as.matrix(Tung)

counts <- counts[1:10000,1:200]
params <- splatEstimate(counts)
```

You can either use the parameters estimated from real data to simulate data or pre-define those parameters by yourself.

```
### Use parameters estimated from real data to simulate data and you only need
### to define the group porportions (group.prob) and path directions (path.from).
sim.SCRIP1 <- SCRIPsimu(data=counts, params=params, method="paths",
                        group.prob = c(0.25, 0.25, 0.25, 0.25),
                        path.from = c(0, 1, 2, 3))

library(scater)
sim1 <- logNormCounts(sim.SCRIP1)
sim1 <- runPCA(sim1)
plotPCA(sim1, colour_by = "Group") + ggtitle("Big DE factors")
```




```
### Predefine parameters to simulate data
sim.SCRIP2 <- SCRIPsimu(data=counts, params=params, method="paths", batchCells=300,
                        group.prob = c(0.25, 0.25, 0.25, 0.25),
                        de.prob = c(0.2, 0.2, 0.2, 0.2),
                        de.downProb = c(0.5, 0.5, 0.5, 0.5),
                        de.facLoc = c(0.2, 0.2, 0.2, 0.2),
                        de.facScale=c(0.2, 0.2, 0.2, 0.2),
                        path.nSteps=c(100, 100, 100, 100),
                        path.from = c(0, 1, 2, 3),
                        path.skew=c(0.5,0.5,0.5,0.5))

library(scater)
sim2 <- logNormCounts(sim.SCRIP2)
sim2 <- runPCA(sim2)
plotPCA(sim2, colour_by = "Group") + ggtitle("Big DE factors")
```


Big DE factors
PCA 2 (2%)
PCA 1 (2%)
5
0
−5
−5
0
5
10
Group
Path1
Path2
Path3
Path4

7 Trajectory DE simulation

To mitigate the side effect of non-significant genes during analysis, it is essential to find genes that are differentially expressed within lineage or between lineages, referred as trajectory DE analysis. SCRIP enables evaluation of trajectory DE analysis methods with simulated DEGs for within lineage or between lineages. Between-lineage and within-lineage DEGs were simulated separately, with multiplicative differential expression factors generated from a log-normal distribution described above.

7.1 Within-lineage trajectory DE simulation

For within-lineage simulation, 20% genes were also randomly selected as within-lineage differentially expressed with other within-lineage DE parameters de.downProb, de.facLoc and de.facScale.

```
library(SCRIP)
library(splatter)
counts <- as.matrix(Tung)

counts <- counts[1:10000,1:200]
params <- splatEstimate((counts))

sim.SCRIP.within <- SCRIPsimu(data=counts, params=params, method="paths",
                              de.prob = 0.2, de.downProb=0.5,
                              de.facLoc=1.2, de.facScale=0.1)
```

```
sim <- sim.SCRIP.within
All.genes <- rownames(counts(sim))
Truepaths <- rowData(sim)
DE.within.path1 <- Truepaths$DEFacPath1
True.within <- rep("common",10000)
True.within[which(DE.within.path1 != 1)] <- "DE"
table(True.within)
```

```
## True.within
## common     DE
##   7940   2060
```

7.1.1 Monocle3

Monocle3 test whether any of the genes change over time in their expression by fitting a generalized linear model.

```
library(monocle3)
expression_matrix <- counts(sim.SCRIP.within)
cell_metadata <- data.frame(libsize = apply(expression_matrix,2,sum))
rownames(cell_metadata) <- colnames(expression_matrix)
gene_annotation <- data.frame(gene_short_name=rownames(expression_matrix))
rownames(gene_annotation) <- rownames(expression_matrix)

cds.within <- new_cell_data_set(expression_matrix,
                                cell_metadata = cell_metadata,
                                gene_metadata = gene_annotation)
```

```
cds.within <- preprocess_cds(cds.within, num_dim = 100, method = "PCA")

cds.within <- reduce_dimension(cds.within,preprocess_method = "PCA",
                        reduction_method = "UMAP")
cds.within <- cluster_cells(cds.within,reduction_method="UMAP")
cds.within <- learn_graph(cds.within)

# We find all the cells that are close to the starting point
closest_vertex <- cds.within@principal_graph_aux[["UMAP"]]$pr_graph_cell_proj_closest_vertex
closest_vertex <- as.matrix(closest_vertex[colnames(cds.within), ])
closest_vertex <- as.numeric(names(which.max(table(closest_vertex))))
mst <- principal_graph(cds.within)$UMAP
root_pr_nodes <- igraph::V(mst)$name[closest_vertex]

# We compute the trajectory
cds.within <- order_cells(cds.within, root_pr_nodes = root_pr_nodes)

# plot_cells(cds.within)

gene_fits <- fit_models(cds.within, model_formula_str = "~pseudotime")
fit_coefs <- coefficient_table(gene_fits)

fit_coefs <- fit_coefs[which(fit_coefs$term != "(Intercept)"),]
head(fit_coefs)
```

```
## # A tibble: 6 x 14
##   gene_short_name num_cells_expre~ gene_id model model_summary status term
##   <chr>                      <int> <chr>   <nam> <named list>  <chr>  <chr>
## 1 Gene1                        138 Gene1   <spe~ <smmry.sp>    OK     pseu~
## 2 Gene2                        113 Gene2   <spe~ <smmry.sp>    OK     pseu~
## 3 Gene3                         60 Gene3   <spe~ <smmry.sp>    OK     pseu~
## 4 Gene4                         35 Gene4   <spe~ <smmry.sp>    OK     pseu~
## 5 Gene5                         62 Gene5   <spe~ <smmry.sp>    OK     pseu~
## 6 Gene6                        120 Gene6   <spe~ <smmry.sp>    OK     pseu~
## # ... with 7 more variables: estimate <dbl>, std_err <dbl>, test_val <dbl>,
## #   p_value <dbl>, normalized_effect <dbl>, model_component <chr>,
## #   q_value <dbl>
```

7.1.2 tradeSeq

tradeSeq implements the associationTest() fucntion to test whether the average gene expression is significantly changing along pseudotime.

```
library(magrittr)
# Get the closest vertice for every cell
y_to_cells <-  principal_graph_aux(cds.within)$UMAP$pr_graph_cell_proj_closest_vertex %>%
  as.data.frame()
y_to_cells$cells <- rownames(y_to_cells)
y_to_cells$Y <- y_to_cells$V1

# Get the root vertices
# It is the same node as above
```

```
root <- cds.within@principal_graph_aux$UMAP$root_pr_nodes

# Get the other endpoints
endpoints <- names(which(igraph::degree(mst) == 1))
endpoints <- endpoints[!endpoints %in% root]

# For each endpoint
cellWeights <- lapply(endpoints, function(endpoint) {
  # We find the path between the endpoint and the root
  path <- igraph::shortest_paths(mst, root, endpoint)$vpath[[1]]
  path <- as.character(path)
  # We find the cells that map along that path
  df <- y_to_cells[y_to_cells$Y %in% path, ]
  df <- data.frame(weights = as.numeric(colnames(cds.within) %in% df$cells))
  colnames(df) <- endpoint
  return(df)
}) %>% do.call(what = 'cbind', args = .) %>%
  as.matrix()
rownames(cellWeights) <- colnames(cds.within)
pseudotime <- matrix(pseudotime(cds.within), ncol = ncol(cellWeights),
                     nrow = ncol(cds.within), byrow = FALSE)

library(tradeSeq)
sce.within <- fitGAM(counts = expression_matrix ,
              pseudotime = pseudotime,
              cellWeights = cellWeights)
```

```
library(tradeSeq)
## Association of gene expression with pseudotime
assoRes <- associationTest(sce.within)
head(assoRes)
```

```
##         waldStat df    pvalue  meanLogFC
## Gene1 0.07409934  4 0.9993304 0.01603664
## Gene2 0.07947961  4 0.9992310 0.01825970
## Gene3 0.85034107  4 0.9315746 0.11427878
## Gene4 0.45795653  4 0.9774627 0.09855867
## Gene5 0.17142236  4 0.9965301 0.04248016
## Gene6 4.16293541  4 0.3844038 0.12874706
```

7.2 Between-lineage trajectory DE simulation

Between-lineage DE factors: Between-lineage DE factors are generated from a log-normal distribution with parameters including number of genes (nGenes), the path-specific proportion of DE genes (de.prob), the proportion of down-regulated DE genes (de.downProb), location factor (de.facLoc) and scale factor (de.facScale).

```
library(SCRIP)
library(splatter)
counts <- as.matrix(Tung)

counts <- counts[1:10000,1:400]
```

```
params <- splatEstimate(counts)

sim.SCRIP.between <- SCRIPsimu(data=counts, params=params, method="paths",
                               group.prob = c(0.5,0.5),
                               de.prob = c(0.0,0.2), de.downProb=c(0.5,0.5),
                               de.facLoc=c(1.2,1.2), de.facScale=c(0.1,0.1),
                               path.from = c(0, 1),path.nSteps = c(100, 100))
```

```
Truepaths <- rowData(sim.SCRIP.between)
DE.between.path <- Truepaths$DEFacPath2
True.between <- rep("common",10000)
True.between[which(DE.between.path != 1)] <- "DE"
table(True.between)
```

```
## True.between
## common     DE
##   7943   2057
```

7.2.1 monocle3

The graph_test() in monocle3 uses a statistic from spatial autocorrelation analysis called Moran's I to identify the DEGs.

```
library(monocle3)
expression_matrix <- counts(sim.SCRIP.between)
cell_metadata <- data.frame(libsize = apply(expression_matrix,2,sum))
rownames(cell_metadata) <- colnames(expression_matrix)
gene_annotation <- data.frame(gene_short_name=rownames(expression_matrix))
rownames(gene_annotation) <- rownames(expression_matrix)

cds <- new_cell_data_set(expression_matrix,
                         cell_metadata = cell_metadata,
                         gene_metadata = gene_annotation)

cds <- preprocess_cds(cds, num_dim = 100, method = "PCA")

cds <- reduce_dimension(cds,preprocess_method = "PCA",
                        reduction_method = "UMAP")
cds <- cluster_cells(cds,reduction_method="UMAP")
cds <- learn_graph(cds,use_partition = FALSE)

# We find all the cells that are close to the starting point
closest_vertex <- cds@principal_graph_aux[["UMAP"]]$pr_graph_cell_proj_closest_vertex
closest_vertex <- as.matrix(closest_vertex[colnames(cds), ])
closest_vertex <- as.numeric(names(which.max(table(closest_vertex))))
mst <- principal_graph(cds)$UMAP
root_pr_nodes <- igraph::V(mst)$name[closest_vertex]

# We compute the trajectory
cds <- order_cells(cds, root_pr_nodes = root_pr_nodes)
plot_cells(cds, color_cells_by = "pseudotime")

pr_graph_test_res <- graph_test(cds, neighbor_graph="knn", cores=8)
```

```
head(pr_graph_test_res)
```

```
##       status    p_value morans_test_statistic     morans_I gene_short_name
## Gene1     OK 0.01841683             2.0876030  0.024823148           Gene1
## Gene2     OK 0.79417286            -0.8209859 -0.013227651           Gene2
## Gene3   FAIL         NA                    NA           NA           Gene3
## Gene4     OK 0.25387387             0.6623488  0.006160813           Gene4
## Gene5     OK 0.05777613             1.5737197  0.018000548           Gene5
## Gene6     OK 0.06451481             1.5179396  0.017261656           Gene6
##         q_value
## Gene1 0.0752791
## Gene2 0.8768830
## Gene3 1.0000000
## Gene4 0.4926497
## Gene5 0.1851635
## Gene6 0.2010128
```

7.2.2 tradeSeq

The function patternTest() implements a statistical model to identify DEGs which have significantly different expression patterns.

```
library(magrittr)
# Get the closest vertice for every cell
y_to_cells <-  principal_graph_aux(cds)$UMAP$pr_graph_cell_proj_closest_vertex %>%
  as.data.frame()
y_to_cells$cells <- rownames(y_to_cells)
y_to_cells$Y <- y_to_cells$V1

# Get the root vertices
root <- cds@principal_graph_aux$UMAP$root_pr_nodes

# Get the other endpoints
endpoints <- names(which(igraph::degree(mst) == 1))
endpoints <- endpoints[!endpoints %in% root]

# For each endpoint
cellWeights <- lapply(endpoints, function(endpoint) {
  # We find the path between the endpoint and the root
  path <- igraph::shortest_paths(mst, root, endpoint)$vpath[[1]]
  path <- as.character(path)
  # We find the cells that map along that path
  df <- y_to_cells[y_to_cells$Y %in% path, ]
  df <- data.frame(weights = as.numeric(colnames(cds) %in% df$cells))
  colnames(df) <- endpoint
  return(df)
}) %>% do.call(what = 'cbind', args = .) %>%
  as.matrix()

rownames(cellWeights) <- colnames(cds)
pseudotime <- matrix(pseudotime(cds), ncol = ncol(cellWeights),
```

```
                    nrow = ncol(cds), byrow = FALSE)

library(tradeSeq)
sce <- fitGAM(counts = expression_matrix,
              pseudotime = pseudotime,
              cellWeights = cellWeights)
```

```
library(tradeSeq)
patternRes <- patternTest(sce)
head(patternRes)
```

```
##           waldStat df    pvalue   fcMedian
## Gene1 3.392170e+00  4 0.4944624 0.20445096
## Gene2 2.803837e+00  2 0.2461243 0.08769774
## Gene3 6.461674e-18  2 1.0000000 0.01444140
## Gene4 1.297625e+00  2 0.5226660 0.08515233
## Gene5 5.249905e+00  4 0.2626031 0.41167851
## Gene6 5.741070e+00  4 0.2193378 0.04577772
```