Differential Expression Analysis of scRNA-seq data using a Simulated dataset

Olympia Hardy

August 2020

Contents

These packages are required to complete the following analysis:

```
library(devtools)
library(Seurat)
library(SingleCellExperiment)
library(SPARSim)
library(splatter)
```

```
library(scDesign)
library(scater)
library(cowplot)
library(dplyr)
library(reshape2)
library(purrr)
library(reticulate)
library(SPsimSeq)
library(RankStat)
library(DESeq2)
library(MAST)
library(DEsingle)
library(aod)
library(arm)
library(fdrtool)
library(lars)
source("/datastore/2505621h/DE_Analysis/Packages/SigEMD-master/FunImpute.R")
source("/datastore/2505621h/DE_Analysis/Packages/SigEMD-master/SigEMDHur.R")
source("/datastore/2505621h/DE_Analysis/Packages/SigEMD-master/SigEMDnonHur.R")
source("/datastore/2505621h/DE_Analysis/Packages/SigEMD-master/plot_sig.R")
library(gtools)
library(Biobase)
library(compcodeR)
```

NOTE: All packages and required dependencies used are freely available to download from CRAN, Bioconductor and Github.

Part 1: Simulator Analysis

Reading in our scRNA-seq data

The data used in this example has been downloaded from the 10X Genomics website where peripheral blood mononuclear cells (PBMC's) were sequenced from a healthy donor using the Chromium Single-Cell 3' v1 platform. You can download the count matrix here.

Firstly we need to read in our raw data that comes in the form of three files:

- matrix.mtx
- barcode.tsv
- features.tsv

Using the `Read10X` function from the `Seurat` package allows us to define the data directory that contains these files and parses them into a sparse Dgs matrix. For our downstream analyses we will need this data to be converted from a sparse matrix, that parses the 0 values in our data as '.' to save on storage, into a standard matrix to be compatible with our simulator packages.

```
pbmc10X <- Read10X(data.dir = "/datastore/2505621h/Simulators/Data/pbmc_filtered_matrices_mex/hg19/")

pbmc10X <- as.matrix(pbmc10X)
```

Now we have our count matrix with each row representing a gene and each column representing the cell:

```
##              AAACATACAACCAC-1 AAACATACACCAGT-1 AAACATACCCGTAA-1
## RP11-54O7.17                0                0                0
## HES4                        0                0                0
## RP11-54O7.11                0                0                0
## ISG15                       2                2                0
## AGRN                        0                0                0
## RP11-54O7.18                0                0                0
```

We will also need our real raw count matrix to be in the format of a `SingleCellExperiment` object for some packages and the final comparison.

```
pbmc10X_sce <- SingleCellExperiment(assays = list(counts = pbmc10X))
```

SPARSim Simulation

Data Normalisation

For the `SPARSim` simulation firstly we need to normalise our real data count matrix. In the `SPARSim` vignette the built in normalisation function taken from the `scran` package `scran_normalization` is defunct. Therefore here we used the `logNormCounts` function taken from the `scater` package.

```
pbmc10X_norm_sce <- logNormCounts(pbmc10X_sce)
pbmc10X_norm <- counts(pbmc10X_norm_sce)
```

Setting Conditions

After the normalisation step we must then define the conditions of our real data. Firstly we must identify which cells belong to a given condition and index the columns accordingly. In this case we only have one condition in our PBMC dataset therefore we can assign all of our columns to condition A.

```
pbmc_condition_A_column_index <- c(1:5419)
```

Once we have indexed our columns to a condition we must then construct a list of all the conditions that will then be passed to estimate the parameters for our simulated data.

```
pbmc10X_conditions <- list(cond_A = pbmc_condition_A_column_index)
```

Parameter Estimation

For parameter estimation of our simulated data `SPARSim` requires three input parameters: our raw count matrix from our real data, our normalised count matrix of the real data and the list of experimental conditions that we defined in the previous step.

```
pbmc10X_parameters <-
  SPARSim_estimate_parameter_from_data(raw_data = pbmc10X,
                                       norm_data = pbmc10X_norm,
                                       conditions = pbmc10X_conditions)
```

Running the SPARSim Simulation

Once we have our estimated parameters derived from the real data we can then run the simulation by using the `SPARSim_simulation` function passing our output object from the previous step. We then get an output of a list of 5 matrices.

```
pbmc10X_result <- SPARSim_simulation(dataset_parameter = pbmc10X_parameters)
```

SPARSim Output

We then get an output of a list of 5 matrices:

- count_matrix
- gene_matrix
- abundance_matrix
- variability_matrix
- batch_factors_matrix

We are interested in the count_matrix which can be extracted from the list using the snippet of code below.

```
pbmc_sparsim <- pbmc10X_result$count_matrix
```

We can then convert the simulated count matrix into a `SingleCellExperiment` object for our downstream comparison.

```
sparsim_pbmc_sce <- SingleCellExperiment(assays = list(counts = pbmc_sparsim))
```

Splatter Simulation

Parameter Estimation

For our next simulator package `Splat` we can pass our real raw data count matrix into the `splatEstimate` function that requires an integer count matrix or a `SingleCellExperiment` object. More details can be found in the `Splatter` vignette here.

This step estimates 5 parameters from the real data: 1. Mean parameters 2. Library size parameters 3. Expression outlier parameters 4. BCV parameters 5. Dropout parameters

```
splat_parameters_pbmc <- splatEstimate(pbmc10X)
```

Splat Simulation

After we have obtained our estimated parameters we can now run the `Splat` simulation that takes the parameters and simulates a count matrix that is output as a `SingleCellExperiment` object.

```
splat_sim_pbmc <- splatSimulate(splat_parameters_pbmc)
```

scDesign Simulation

The `scDesign` simulator package contains 3 main functions which can be found in detail here. In our analysis we will use the `design_data` function that allows us to simulate a raw count matrix based on a real scRNA-seq dataset. In a straightforward example where there is only one group in our data (`ngroup = 1`) the only arguments we need to define is the number of cells denoted by `ncell` and the total read number per cell defined as `S`. In the event that there is more than one group we have to define additional arguments:

- `pUp` : Proportion of upregulated genes
- `pDown` : Proportion of downregulated genes
- `fU` : Upper bound of fold changes of DE genes
- `fL` : Lower bound of fold changes of DE genes

The output of this simulation is a large integer matrix so we must parse the output to a `SingleCellExperiment` object for our comparison.

```
scDesign_pbmc <- design_data(pbmc10X, S = 1e+08, ncell = 5419, ngroup = 1)
scDesign_pbmc_sce <- SingleCellExperiment(assays = list(counts = scDesign_pbmc))
```

SPsimseq Simulation

`SPsimseq` requires a `SingleCellExperiment` object as an input and carries out a density estimation to estimate the distribution of gene expression from real data to then simulate a count matrix. A more detailed despcription can be found at the Github repository of the package here. It takes a number of arguments that defines the properties of our simulated data:

- `n.sim` : The number of simulated datasets it will output
- `s.data` : A SingleCellExperiment object of our real data count matrix
- `n.genes` : Number of genes we want to be simulated
- `batch.config` : Our data is from a single batch so is equal to 1
- `group.config` : The number of groups we have in our data that must equal to 1
- `tot.samples` : The number of cells we have in our data
- `pDE` : Percentage of DE genes
- `lfc.thrld` : The log fold threshold that defines DE genes
- `model.zero.prob` : The zeroes in our data will be modelled separately when equal to `TRUE`
- `result.format` : We define the output format of our simulated count matrix

```
spsim_pbmc_sim <- SPsimSeq(n.sim = 1,
                           s.data = real_pbmc_sce,
                           n.genes = 32738,
                           batch.config = 1,
                           group.config = 1,
                           tot.samples = 5419,
                           pDE = 0.1,
                           lfc.thrld = 0.25,
                           t.thrld = 1.5,
                           model.zero.prob = TRUE,
                           result.format = "SCE")
```

The output of the simulation is a list of 3 objects with the first being the `SingleCellExperiment` object that contains our count matrix. This can be extracted from the list using the snippet of code below.

```
spsim_pbmc_sce <- spsim_pbmc_sim[[1]]
```

Comparison of Simulator Packages

To compare the performance of our 4 simulator packages against our real data we can use the `compareSCEs` and `diffSCEs` functionalities in the `Splatter` package that takes a list of `SingleCellExperiment` objects. The output from these analyses is a list of 3 objects:

- A list of 8 plots generated by ggplot2
  - Distribution of means
  - Distribution of variance
  - Mean-Variance relationship
  - Library sizes
  - Sparsity by gene
  - Sparsity by cell
  - Mean-Sparsity relationship
  - Variance-Gene correlation
- List containing information about the `colData` of our comparison
- List containing information about the `rowData` of our comparison

The `compareSCEs` plots all `SingleCellExperiment` objects on one graph per metric to give a direct comparison of how similar the simulated dataset is to the real data.

```
comparison_pbmc <- compareSCEs(list(Real = real_pbmc_sce,
                                    Splat = splat_pbmc_sce,
                                    SPARSim = sparsim_pbmc_sce,
                                    scDesign = scDesign_pbmc_sce,
                                    SPsimSeq = spsim_pbmc_sce))
```

The `diffSCEs` function takes the same list of `SingleCellExperiment` objects however the real dataset is defined as the reference. When plotted the real data is represented by a red line that intercepts the x-axis at 0 and the other simulated datasets are plotted against this line to observe how different they are to the real data.

```
diff_comp_pbmc <- splatter::diffSCEs(list(Real = real_pbmc_sce,
                                    Splat = splat_pbmc_sce,
                                    SPARSim = sparsim_pbmc_sce,
                                    scDesign = scDesign_pbmc_sce,
                                    SPsimSeq = spsim_pbmc_sce),
                                    ref = 'Real')
```

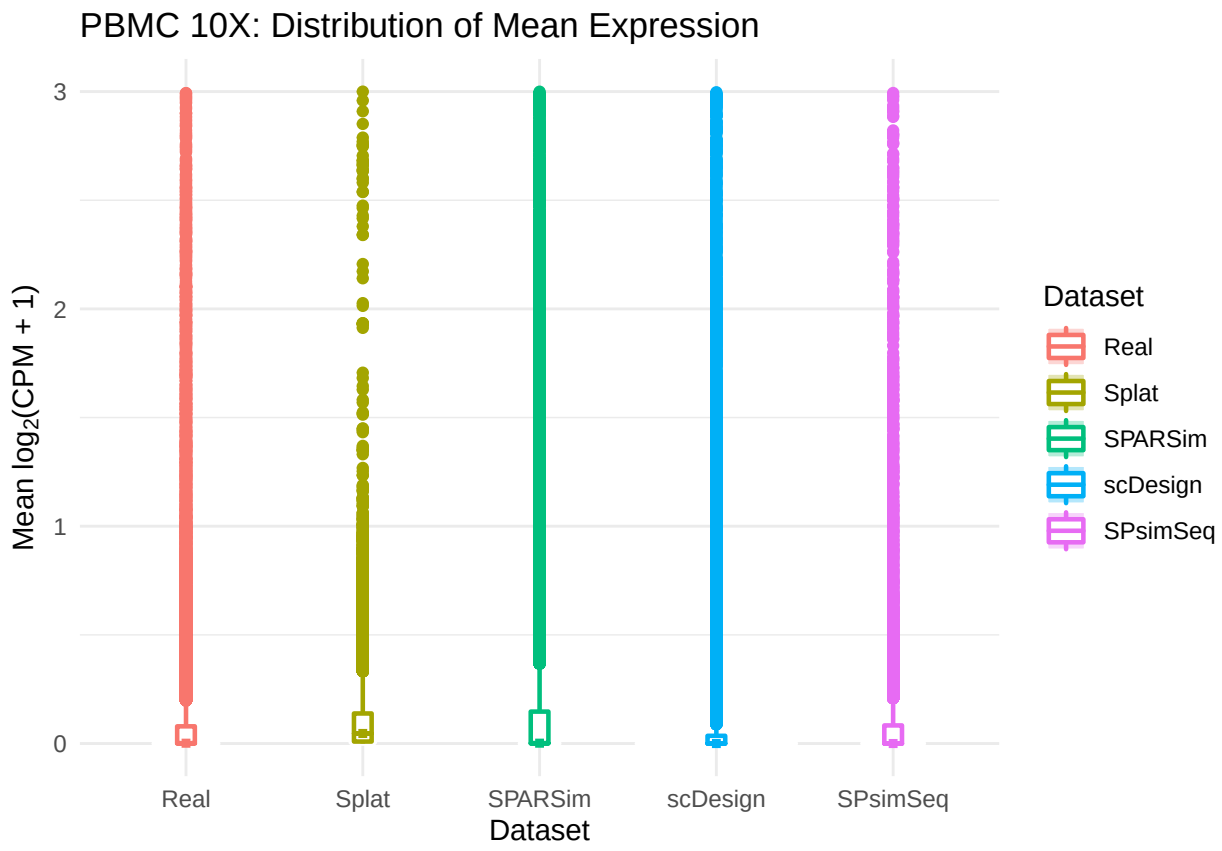
Results

Here we are interested in the plots, we can view the names of the various plots generated using the snippet of code below.

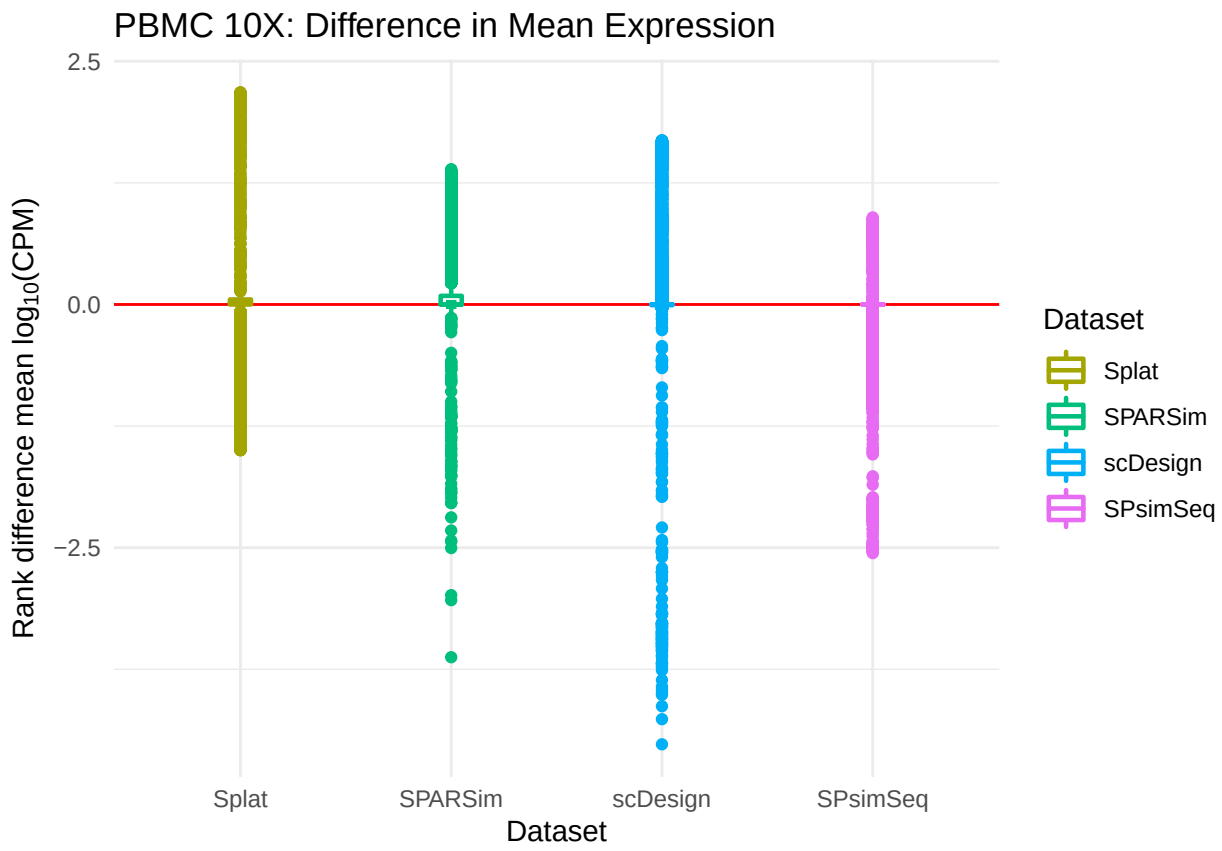
```
names(comparison_pbmc$Plots)
```

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

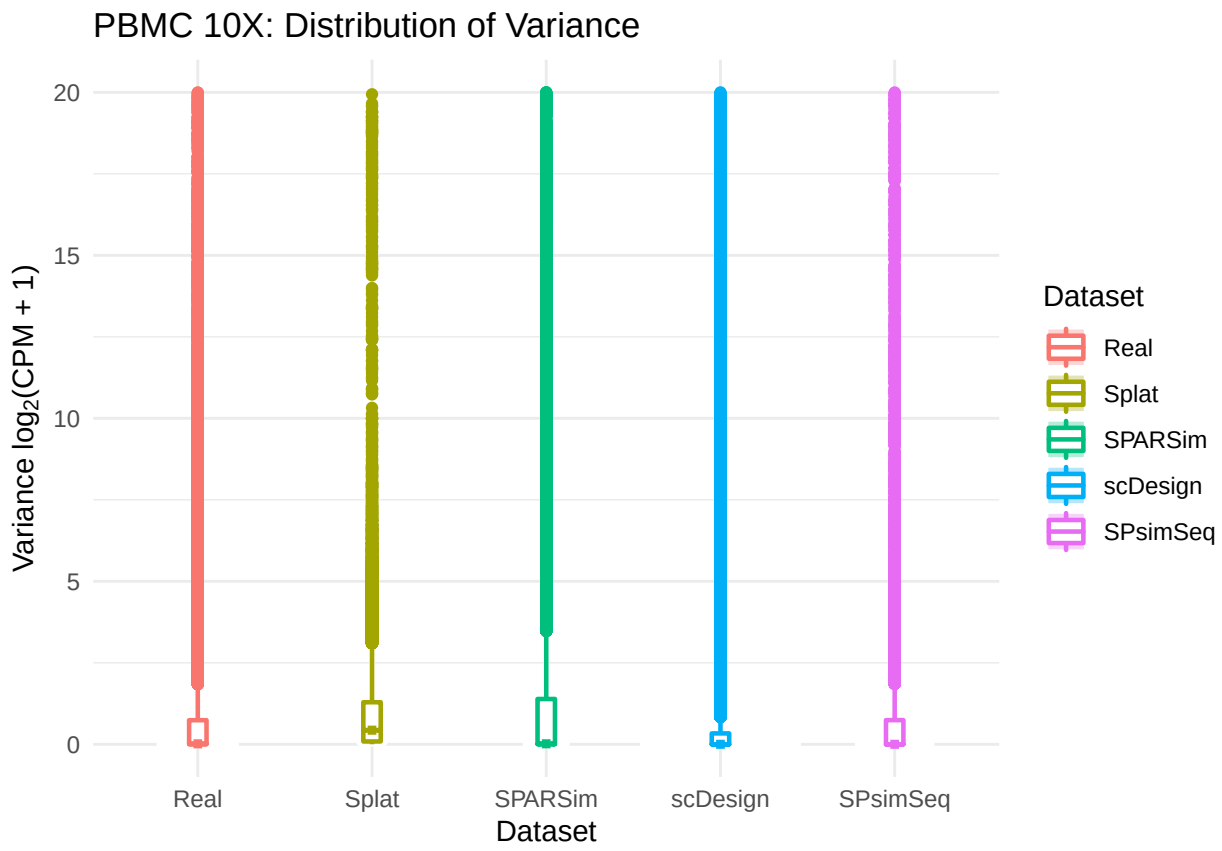
```
mean_plot_pbmc
```



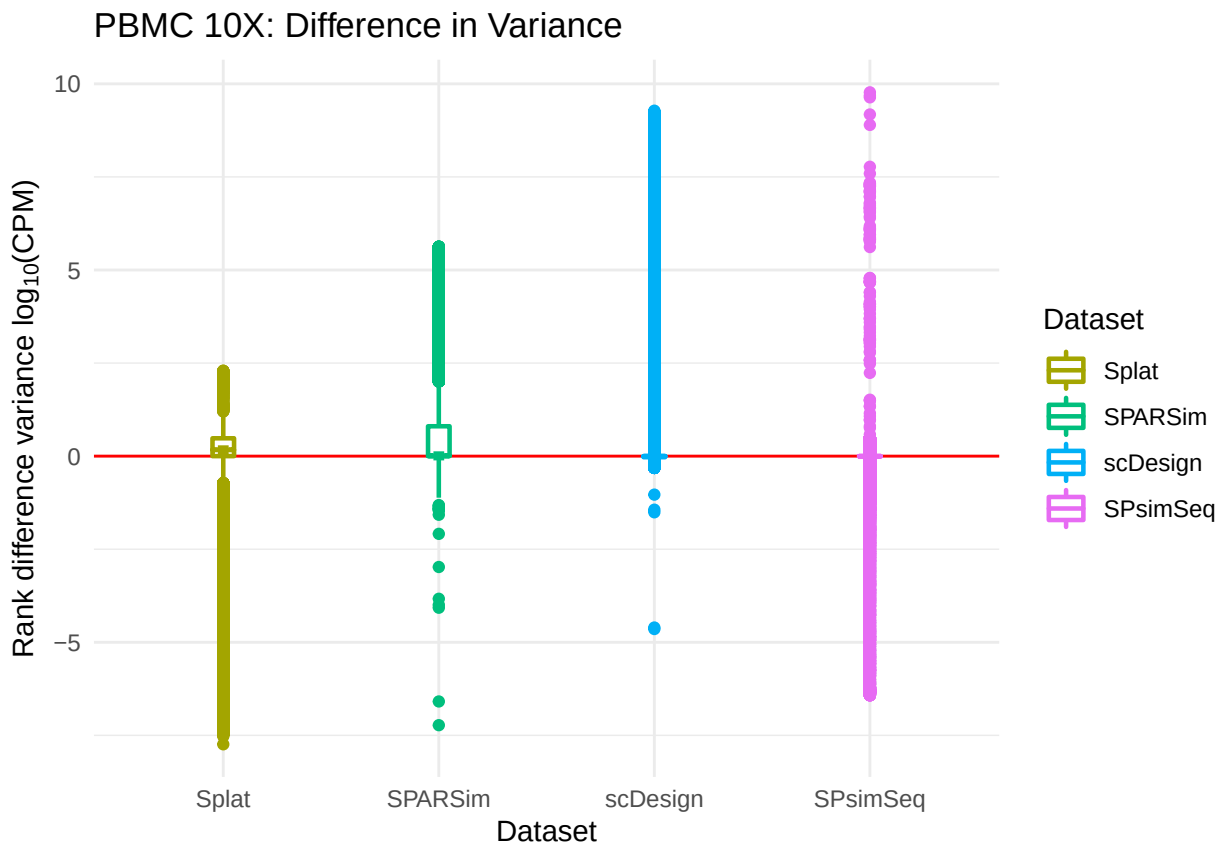

```
diff_mean_plot_pbmc
```



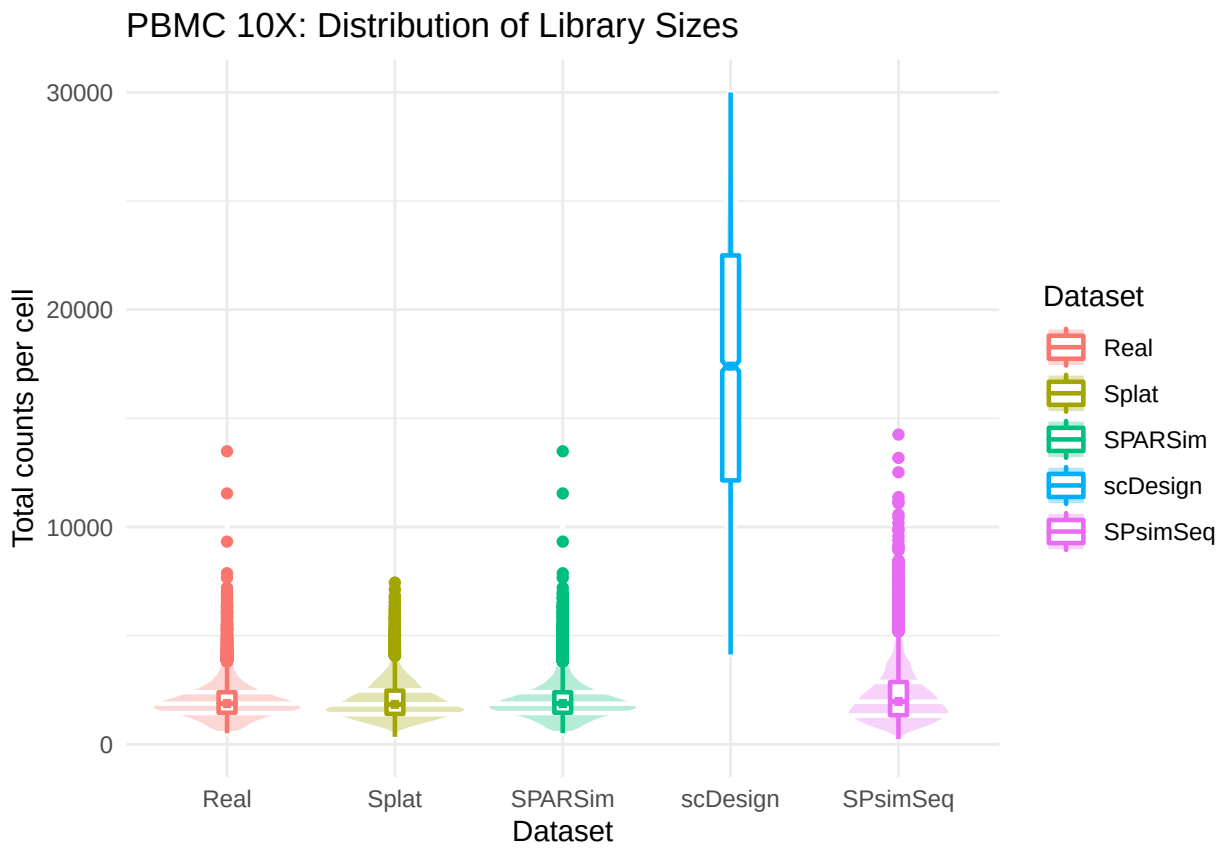

```
variance_plot_pbmc
```



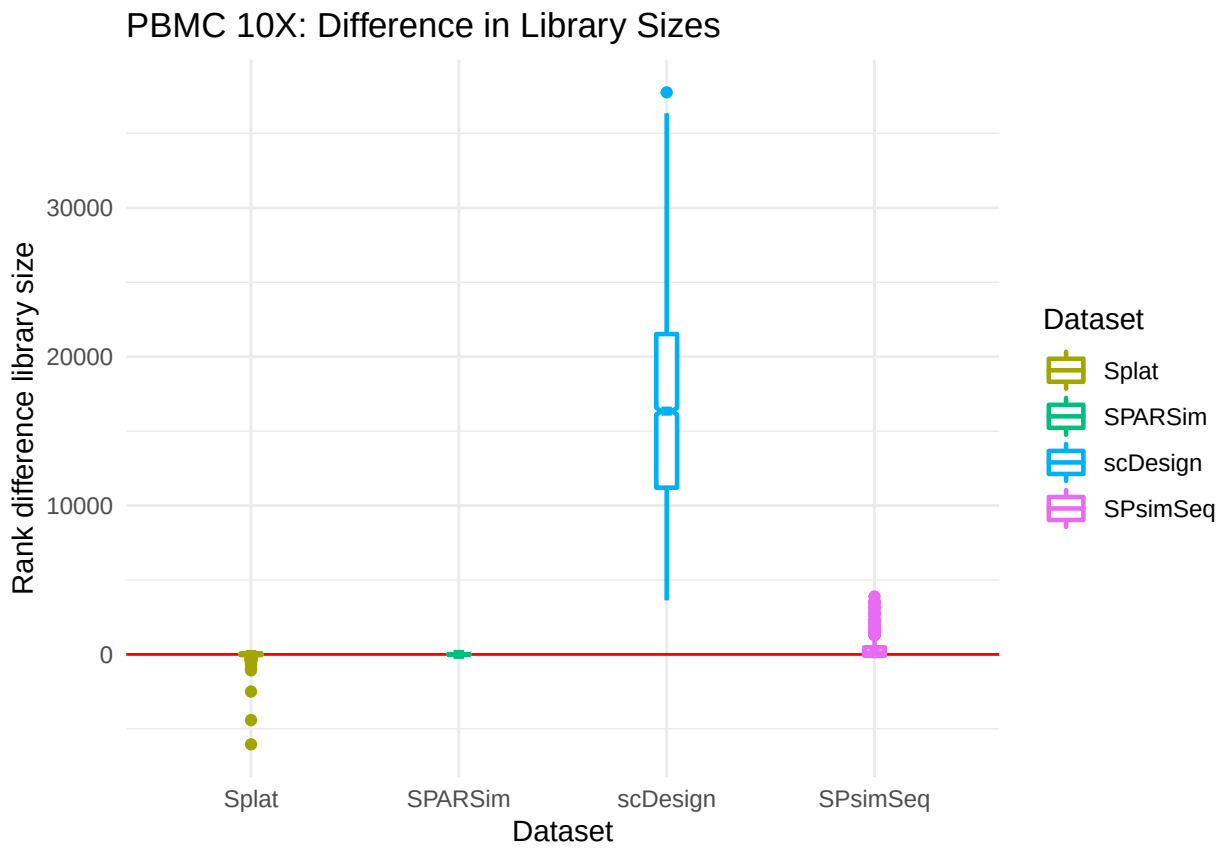

```
diff_variance_plot_pbmc
```



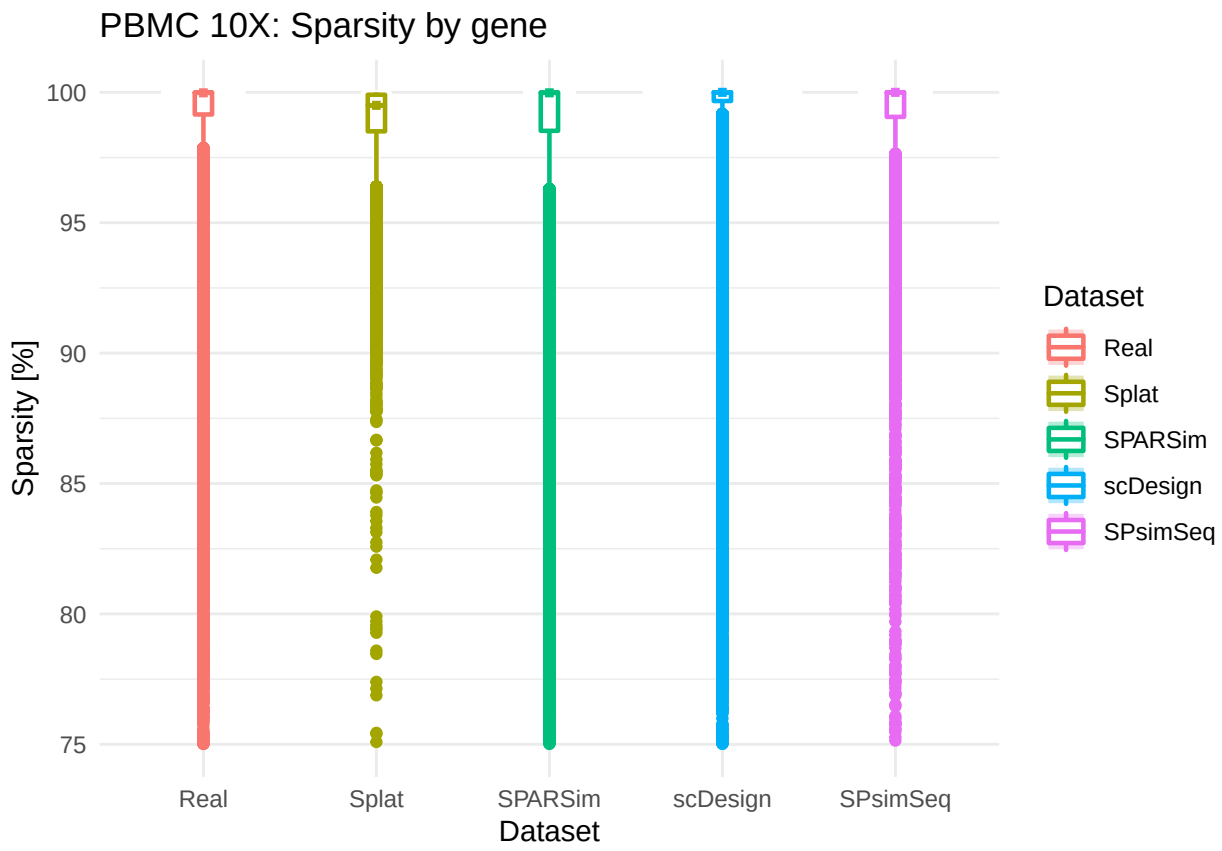

```
library_size_pbmc
```



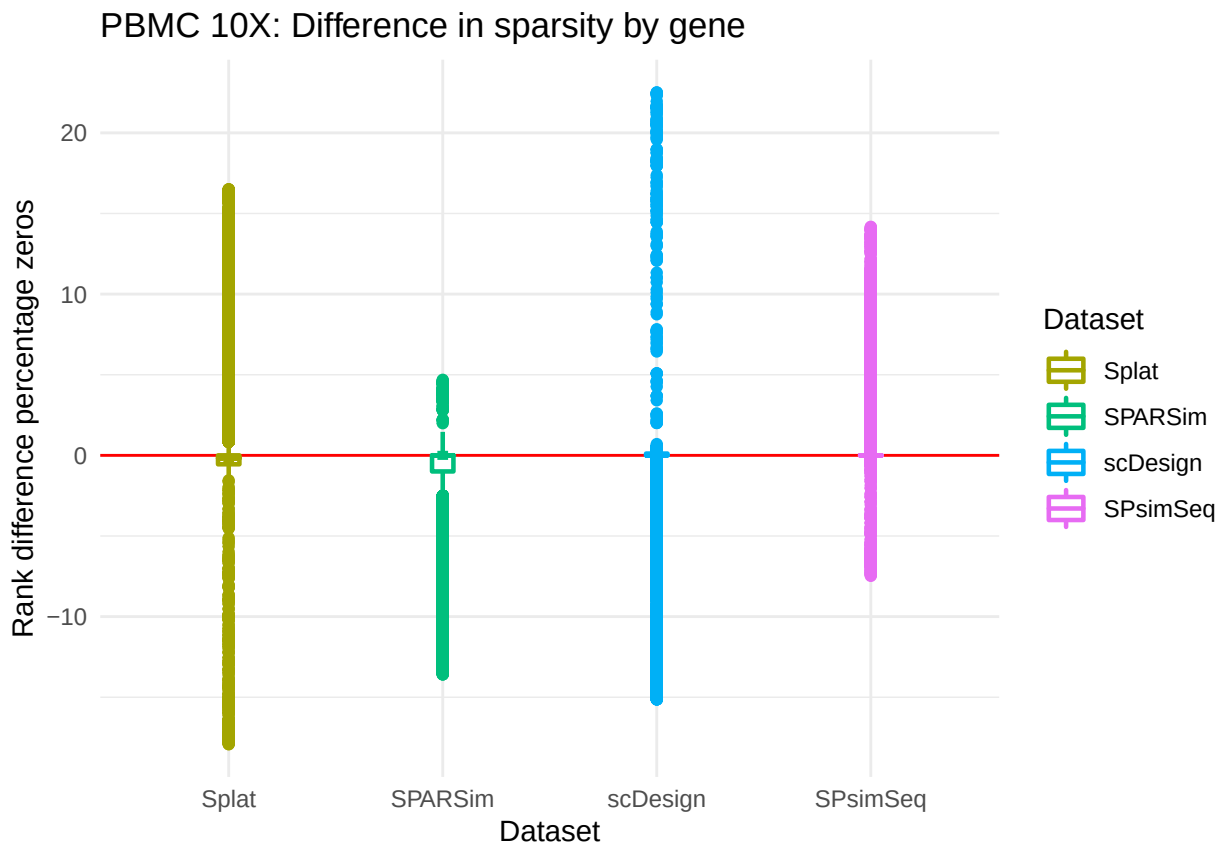

```
diff_library_size_pbmc
```



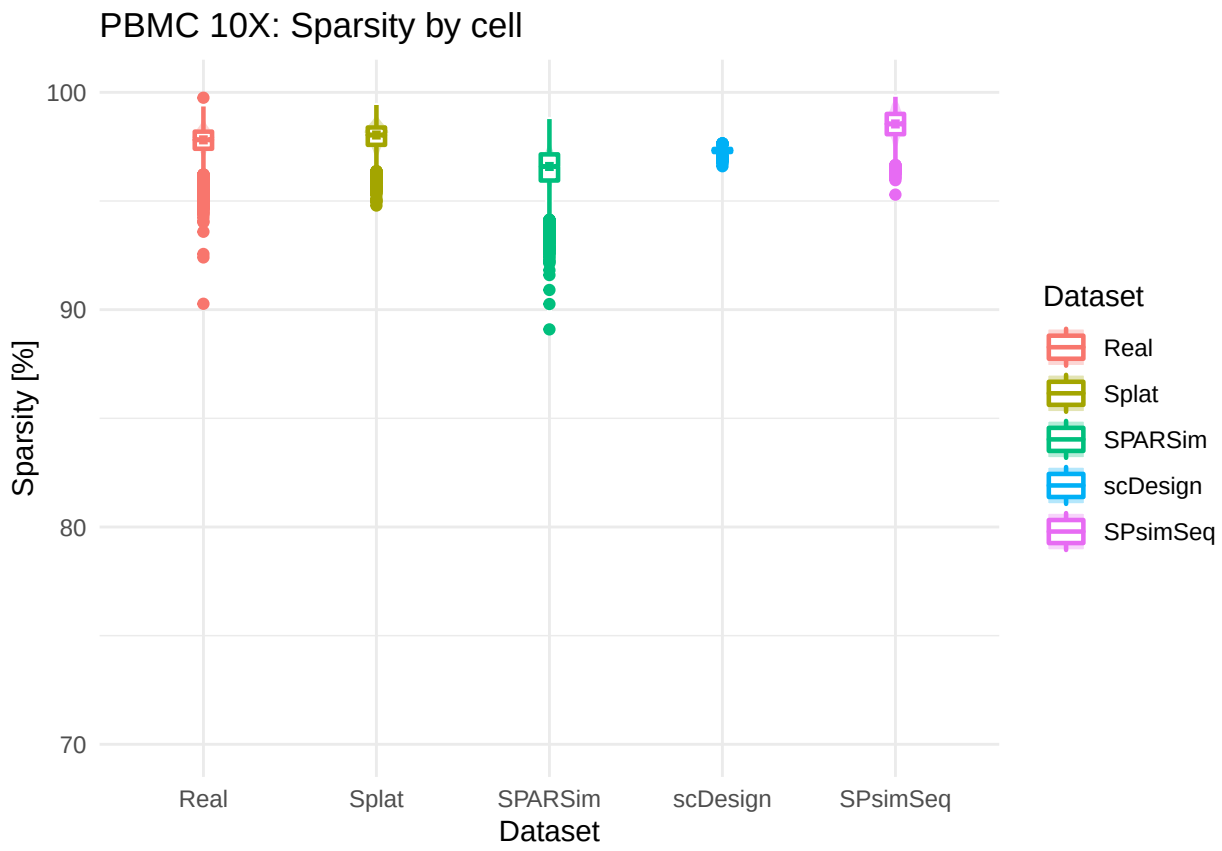

```
zero_gene_plot_pbmc
```



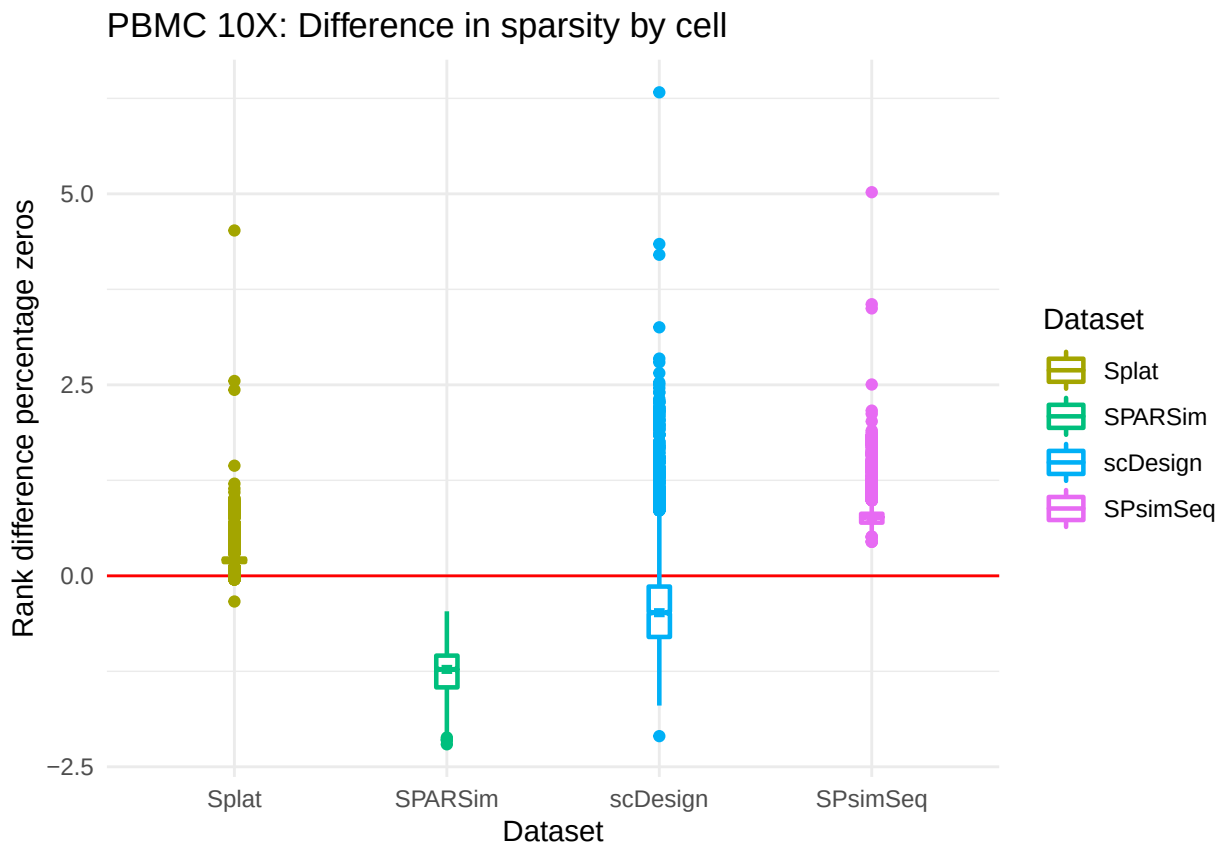

```
diff_zero_gene_plot_pbmc
```



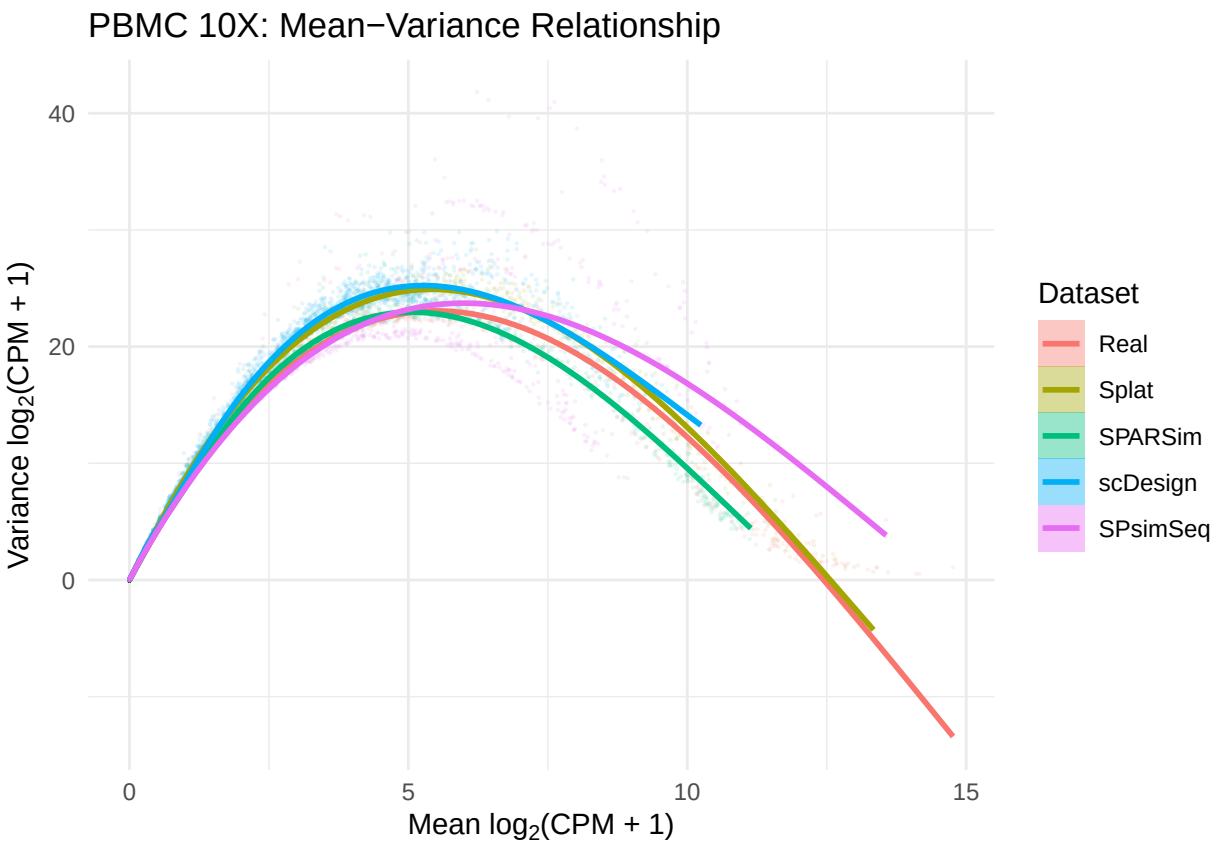

```
zero_cell_plot_pbmc
```



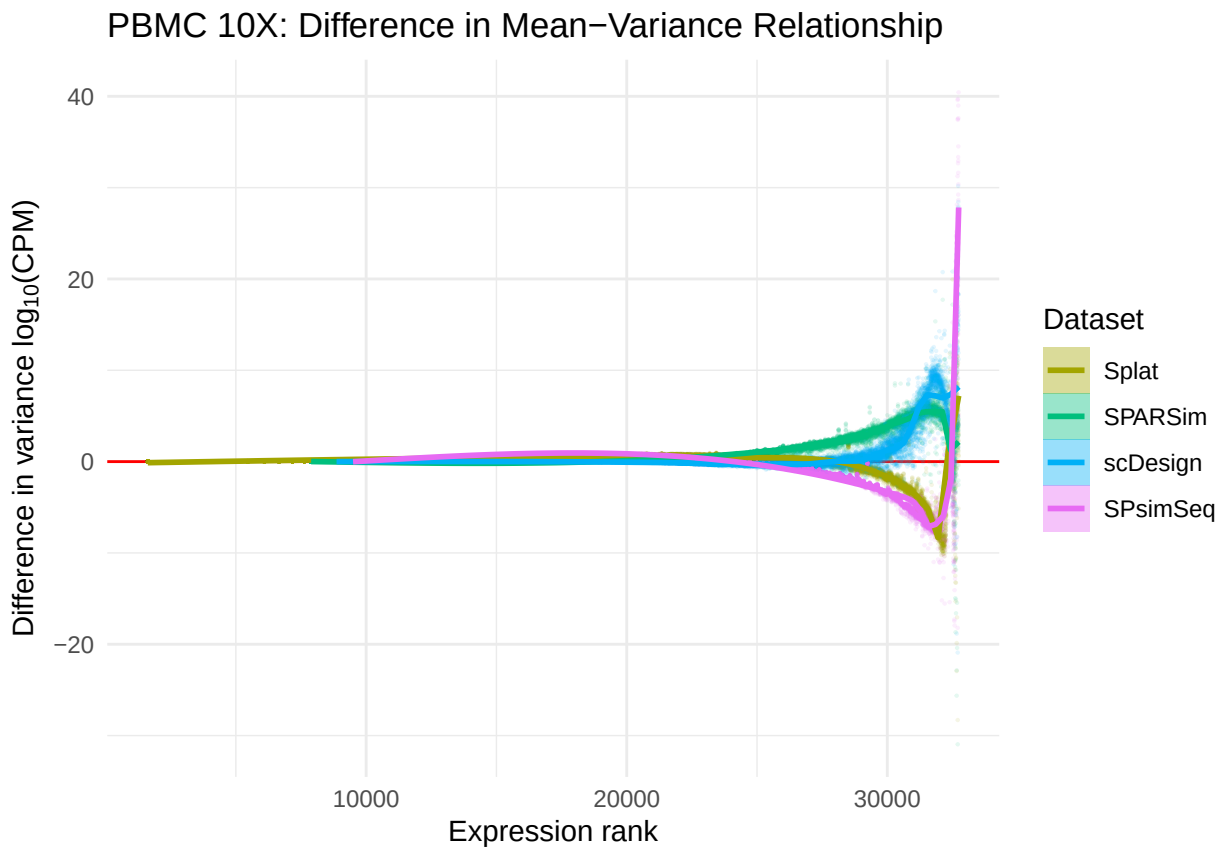

```
diff_zero_cell_plot_pbmc
```



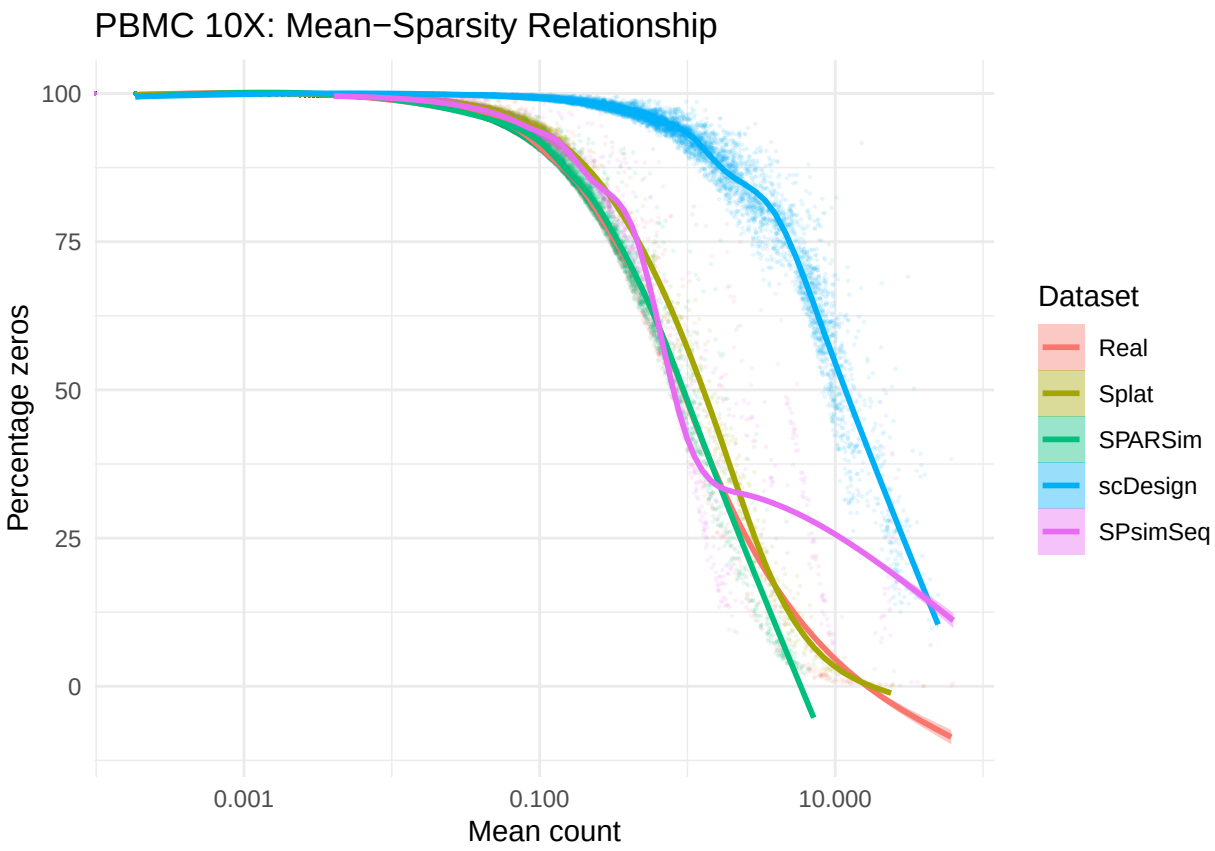

```
mean_variance_plot_pbmc
```



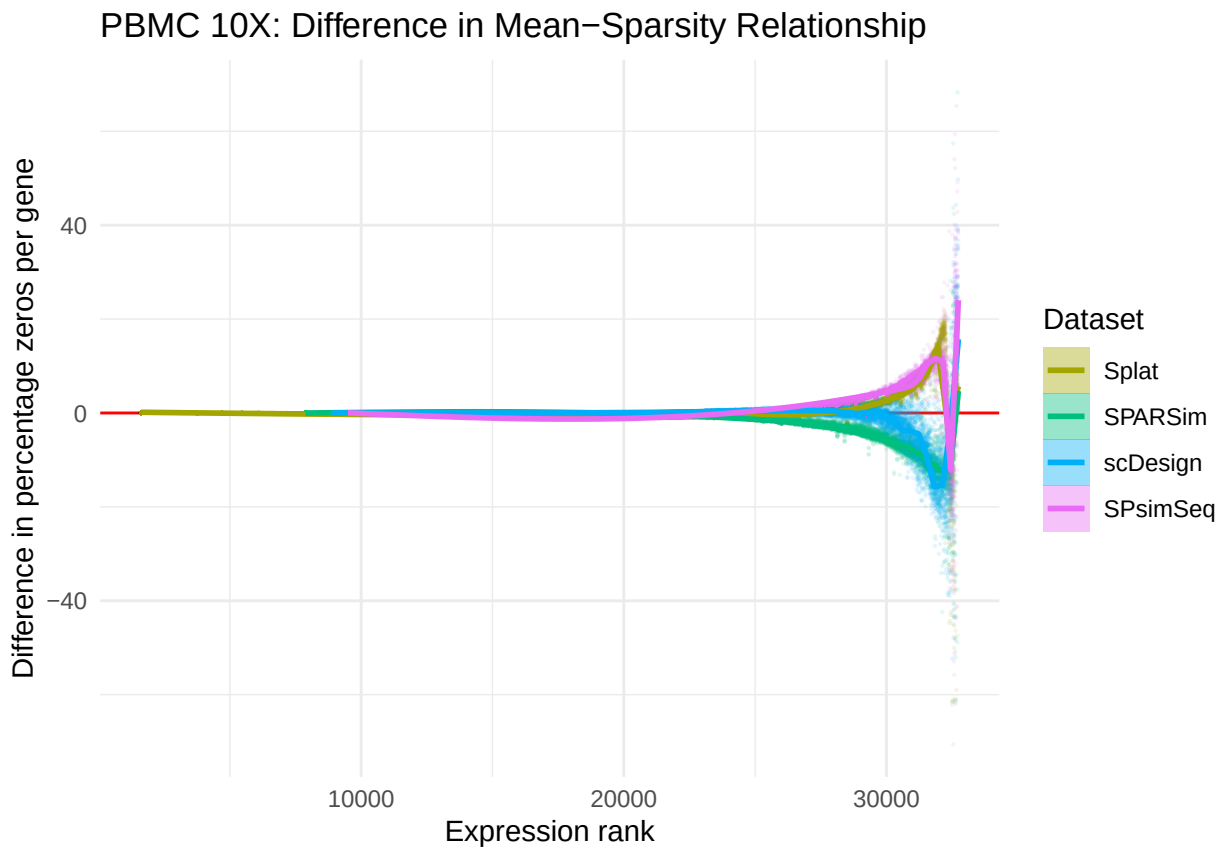

```
diff_mean_variance_plot_pbmc
```




```
mean_zeros_plot_pbmc
```




```
diff_mean_zeros_plot_pbmc
```



Simulator Performance Heatmap

The performance of scRNA-seq simulators can rarely be evaluated using only one dataset. This is because of the variation in scRNA-seq datasets that may come from either technical variation such as 'drop-out' events or biological variation for example multimodal gene expression between sub-populations of cells. A useful way to summarise the performance of data simulators across multiple datasets would be to compile the results into a ranked heatmap. The code used to produce this figure has been taken from the Github repository from the Splatter paper (Zapia et al. 2017) and can be found here. In the example below the analysis detailed above has been carried out on 7 different 10X datasets including the PBMC 10X.

We will be using our 'difference from the real data' objects that we obtained using the `diffSCEs` function from the `Splatter` package. Using `summariseDiff` from `Splatter` the MAD statistic is obtained by calculating the mean expression values from the real data and each of the simulated datasets. These values are sorted, and the real expression value is subtracted from the simulated expression value. The median of the difference between the two values is then used as the MAD statistic. These MAD values are then assigned a ranking (MADRank) between 1-4 with 1 being the most representative of the real data and 4 being the least.

```
##   Simulator Statistic         MAD  MADScaled MADRank Dataset
## 1     Splat      Mean 0.033736943  1.4988811     4.0    PBMC
## 2   SPARSim      Mean 0.001364280 -0.4450127     3.0    PBMC
## 3  scDesign      Mean 0.000000000 -0.5269342     1.5    PBMC
## 4  SPsimSeq      Mean 0.000000000 -0.5269342     1.5    PBMC
## 5     Splat  Variance 0.309353837  1.4992889     4.0    PBMC
## 6   SPARSim  Variance 0.009998291 -0.4562211     3.0    PBMC
```

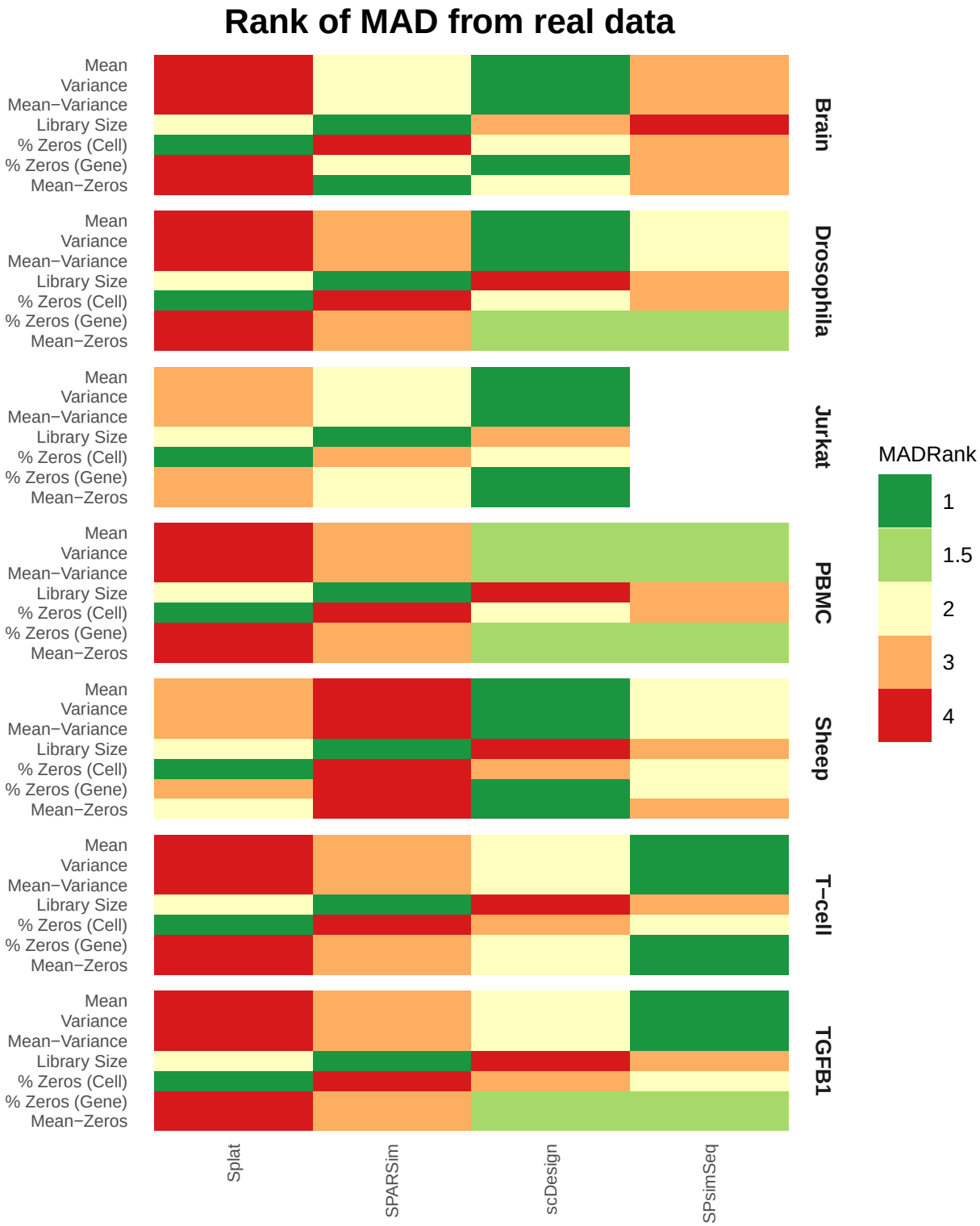
Note: The summary table given by `summariseDiff` requires a bit of wrangling to result in the table above. The first column containing the simulators is called 'Dataset' which must be renamed. Also for the heatmap

to have the correct labels we must add an additional column denoting the dataset the values belong to. This can be done using the code below:

```
names(summary_pbmc)[1] <- "Simulator"
Dataset <- rep("PBMC", length(summary_pbmc))
summary_pbmc$Dataset <- Dataset
```

When this has been done for each dataset we want to compare we must create a master summary table to plot the heatmap. This can be done by joining each dataframe using `rbind`.

```
plot
```


Rank of MAD from real data
Mean
Variance
Mean−Variance
Library Size
% Zeros (Cell)
% Zeros (Gene)
Mean−Zeros
Brain
Drosophila
Jurkat
PBMC
Sheep
T−cell
TGFB1
Splat
SPARSim
scDesign
SPsimSeq
MADRank
1
1.5
2
3
4

Part 2: Differential Analysis using a synthetic dataset

As we can see from the MAD heatmap, `scDesign` was the overall best data simulator package we tested out of the real scRNA-seq datasets we used. For this part of the analysis pipeline we will be using the Drosophila dataset to estimate parameters from for our synthetic dataset. Here we tested nine different differential expression analysis packages and evaluated their performance using the `compcodeR` package:

- DESeq2
- DEsingle
- MAST
- SigEMD
- RankStat methods:
  - Rank Product
  - Rank Sum
  - Rank Distance
  - Reverse Rank Distance
  - Differential Rank Product

To carry out the differential expression analysis we will simulate a two-condition problem and identify which packages are the best at detecting truly differentially expressed genes across the two groups of cells. We will estimate the parameters for our count matrix from the real Drosophila dataset mentioned and define 10% of the genes in the data to be differentially expressed. `scDesign` will tell us which genes in our data are truly differentially expressed so therefore we can compare this ground truth list with the results returned by the DE packages and evaluate their performance.

Data Simulation for DE Analysis

To simulate two groups of cells we must change `ngroup` and for the other parameters required by `scDesign` we must split the cells into two repetitions. Below shows the snippet of code to simulate 50 cells in each group with 5% of the genes in the data to be up regulated and 5% of the genes to be downregulated.

```
scDesign_drosophila <- scDesign::design_data(realcount = drosophila10X, S = rep(1e7, 2), ncell = rep(50
                                             pUp = 0.05, pDown = 0.05, ncores = 1)
```

This gives you a list of three objects:

- List of two count matrices
  - The count matrix for the first group of cells
  - The count matrix for the second group of cells
- A character matrix of up-regulated gene names
- A character matrix of down-regulated gene names

To obtain our master count matrix we must combine both count matrices for the two groups of cells which can be done using `cbind`.

```
drosophila <- cbind(scDesign_drosophila$count$count1, scDesign_drosophila$count$count2)
```

To create our final count matrix ready for DE analysis we must first filter our data for genes that show expression in none of our cells. This can be done using the `dataclean` function from the `SigEMD` package

which removes rows that are 0 across all columns. By carrying out this filtering step it ensures that we are only looking at genes that have expression in our cells. Some packages do not have a built-in filtering functionality and if there are rows that are all 0 the function will throw an error. Also we need to make sure all the result matrices we get from our DE packages are the same dimensions otherwise the comparison in `compcodeR` will not work.

```
drosophila <- dataclean(drosophila)
```

```
## Remove genes that all are zeros...

## done
```

Now we have our matrix with expressed arbitrary genes as row names and our column names are labelled with the group and number (e.g. Cell 1 in Group 2 = C2_1). The first 5 cells of each group are shown below:

```
##        C1_1 C1_2 C1_3 C1_4 C1_5 C2_1 C2_2 C2_3 C2_4 C2_5 C2_6
## gene1     0    0    0    0   40    0  129    0   70    0    0
## gene10    0    0    0    0    0  144    0    0    0    0    0
## gene12    0    0    0    0    0    0   56   13 1019   71  403
## gene14   28    0    0    0   97    0    0    0    0    0    0
## gene16    0    0    0    0   32 1126  235   38  660  100  747
## gene25    0    0    0    0    0    0    0    0    0    0    0
```

Creation of DE gene annotation

When we create our ground truth list for it to be used in `compcodeR` we must assign each gene a binary classification to identify whether the gene is DE or non-DE. First we need to extract the names of the genes that are up- and down-regulated from the `scDesign` output and combine them into a matrix of DE genes

```
genesUp <- scDesign_drosophila[[2]]
genesUp <- genesUp[[2]]

genesDown <- scDesign_drosophila[[3]]
genesDown <- genesDown[[2]]

DE_genes <- as.character(append(genesUp, genesDown))
```

Next we find out which genes are non-DE by creating a matrix with all the names in our synthetic dataset and writing the gene names to a new maxtrix 'non-DE' that are not found in our DE genes matrix

```
genes <- row.names(drosophila)
nonDE_genes <- setdiff(genes, DE_genes)
```

Now we can create our ground truth as a list of two dataframes one containing names of DE genes and the other non-DE

```
ground_truth <- list(
  DE = as.data.frame(DE_genes),
  nonDE = as.data.frame(nonDE_genes)
)
```

We can assign a binary annotation for our genes with 0 denoting a non-DE gene and 1 representing a DE gene. We can create a dataframe with the names of the genes in our data and add a column called 'DE_State' to contain our binary notation. First assign 0 to all the genes in the dataframe and then assign a 1 to the genes that are found in our DE genes list:

```
genes <- as.data.frame(row.names(drosophila))
genes$DE_State <- rep(0, times = length(genes))
genes$DE_State[genes$`row.names(drosophila)` %in% ground_truth$DE$DE_genes] = 1
```

For the `compcodeR` analysis we need to do a little wrangling to get our annotation to the correct format. The names of the genes must be the rownames and the only column should be the DE_State classification that **must** be named 'differential.expression':

```
rownames(genes) <- genes[,1]
genes[,1] <- NULL
DE_State_annotation <- genes
names(DE_State_annotation)[1] <- "differential.expression"
```

Now we have our simulated dataset and our DE gene annotation matrix we can proceed to running the differential analysis packages.

MAST

The vignette for the Model-based Analysis of Single-cell Transcriptomics (`MAST`) can be found here. The method uses an input of log transformed transcripts per million (TPM) +1 values so therefore we must convert our count matrix. This was done by converting the count matrix into an `ExpressionSet` matrix and using a default estimation of 1e6 transcripts per cell.

```
eset_drosophila <- ExpressionSet(drosophila)
counts = exprs(eset_drosophila)
tpm <- counts*1e6/colSums(counts)
tpm <- log2(tpm+1)
```

Next we must supply `MAST` with a condition matrix so that it knows which cells in our data belong to which group

```
coldata_MAST <- data.frame(
  condition = factor(c(
    rep("0", 50),
    rep("1", 50))))

row.names(coldata_MAST) <- colnames(drosophila)
```

`MAST` also calculates a cellular detection rate (CDR) and uses it as a co-variate of the data as well as the condition matrix. The following code is taken from the vignette example from `MAST`. First we must convert our log TPM+1 matrix into a `SingleCellAssay` object and then we can calculate the CDR:

```
sca <- FromMatrix(tpm,  cData = coldata_MAST)
cdr <- apply(exprs(eset_drosophila),2,function(x) mean(x>0))
col_D <- colData(sca)
col_D$cdr <- cdr
col_D$cdr <- CD$cdr-mean(cdr)
colData(sca) <- col_D
```

Next we can run the differential analysis test followed by a likelihood ratio test between the two conditions.

```
testing <- zlm(~ condition + cngeneson, sca = sca)

summaryCond_test <- summary(testing, doLRT='condition1')
```

Then we extract the results table and the relevant information we need to identify DE genes such as the caluclated p-values and the FDR.

```
summaryDt <- summaryCond_test$datatable

fcHurdle <- merge(summaryDt[contrast=='condition1' & component=='H',.(primerid, `Pr(>Chisq)`)],
                  summaryDt[contrast=='condition1' & component=='logFC', .(primerid, coef, ci.hi, ci.lo
fcHurdle[,fdr:=p.adjust(`Pr(>Chisq)`, 'fdr')]
```

DEsingle

`DEsingle` is a package derived from the `DESeq2` method that is optimised to analyse scRNA-seq data. More detail of this can be found in the corresponding paper. To begin the analysis we must first define our group matrix:

```
group <- factor(c(
  rep("1", 50),
  rep("2", 50)))
```

Then we can run the analysis on our data using the code below. The output of the package provides a functionality to split the DE genes into three distinct groups at a specified threshold:

- DEs
- DEa
- DEg

More information on this is included in the vignette, for this analysis this is not important as we only wanted to identify DE genes regardless of type however the functionality is also shown in the code below:

```
deSingle_drosoph <- DEsingle(drosophila, group = group)

dros_result_groups <- DEtype(results = deSingle_drosoph, threshold = 0.05)
```

SigEMD

The vignette for `SigEMD` can be found here. This is not a package like the other ones used in this tutorial that can be loaded from `library` but rather the functionalities must be read into the Global Environment in R to then be used as normal. Here we create our condition matrix as normal and we will use our TPM+1 matrix that we generated from our `MAST` analysis. However, SigEMD can be compatible with any pre-normalised data as input but be aware the package lacks an internal normalisation function so this step must be done before the analysis. We can then run the analysis using 100 permutations to obtain more accurate p-values as recommended in the vignette.

```
condition_sigEMD <- c(
  rep("0", 50),
  rep("1", 50))
names(condition_sigEMD) <- colnames(tpm)

results_EMD <- calculate_single(data =  data,condition =  condition_sigEMD, Hur_gene = NULL, binSize=0.1

emd<- as.data.frame(results_EMD$emdall)
```

RankStat

The `RankStat` package contain five non-parametric methods Rank Products (RP), Rank Sum (RS), Rank Distance (RD), Reverse Rank Distance (RRD) and Differential Rank Products (DRP). The methods are all a derivative of the Rank Product method described in this paper. This method requires regularised log transformed values as input (rlog+1) . We can transform our count matrix using the `rlogTransformation` function from the `DESeq2` package. It is worth mentioning that for sample sizes larger than 50 this step takes a long time to run.

```
# RankStat Analysis

drosophila_rlog <- DESeq2::rlogTransformation(drosophila + 1)
drosophila_rlog <- as.data.frame(drosophila_rlog)

drosophila_rlog <- dataclean(drosophila_rlog)
```

Before we run the `RankStat` analysis we must first create a dataframe that contains all the gene names that we have in our data, followed by creating a vector denoting which cells belong to a given group.

```
gene_names <- row.names(drosophila)

coldata <- data.frame(
  condition = factor(c(
    rep("0", 50),
    rep("1", 50))))

coldata$condition <- relevel(coldata$condition, ref = "0")

coldata <- as.vector(t(coldata))
```

Now that we have our annotations ready we can conduct a two-sided test that will calculate p-values and rank them accordingly. The "two.sided" test argument transforms the results of two separate one-sided tests and sorts the genes without detailing what direction the change is occuring making this a useful option for determining marker genes across two conditions.

```
result_RS <- RankStat(drosophila_rlog, cl = coldata, method = c("RP", "DRP", "RS", "RD", "R_RD"), alter
```

The output from this analysis will give you:

- A table of fold-change expressions between the two conditions
- A dataframe with the results of the two-sided test
- A list of the permutation types for the data

- A list of the class type which contains labels for which group the cells belong to

Here we are interested in the dataframe of results for each method which can be extracted using the code below:

```
result_RP <- result_RS$RP_2s
result_DRP <- result_RS$DRP_2s
result_RSum <- result_RS$RS_2s
result_RD <- result_RS$RD_2s
result_R_RD <- result_RS$R_RD_2s
```

compcodeR

To evalute and compare the results of each differential expression package we used the `compcodeR` package which has superb documentation that can be found here. The package is mainly designed for bulk RNA-seq data analysis and contains wrappers to carry out differential expression from additional methods such as `DESeq2`, `edgeR` and `NOISeq`. However it does allow the flexibility to provide your own differential analysis result data given that it is correctly coerced into a `compcodeR` object. In this analysis we carried out the `DESeq2` analysis on our scRNA-seq data within `compcodeR` and then formatted the results of the rest of the analyses into `compcodeR` objects for comparison.

To create a `compcodeR` data object we must supply a sample annotation table which shows which cells belong to which group with cell names as the row names of the dataframe.

```
coldata_compcodeR <- data.frame(
  condition = factor(c(
    rep("0", 50),
    rep("1", 50))))

row.names(coldata_compcodeR) <- colnames(drosophila)
```

We must also then create the parameters that contain information about are data such as:

- The name of the dataset
- The number of samples per condition
- The number of truly differentially expressed genes in our data
- The fraction of DE genes which are upregulated
- A unique 10-digit dataset identification number

```
parameters <- list(dataset = "Drosophila",
                   samples.per.cond = 50,
                   n.diffexp = 1760,
                   fraction.upregulated = 0.5,
                   uID = 1234567890)
```

We must also supply the object with an annotation matrix with our binary classification of DE and non-DE genes that we created at the beginning of the analysis:

```
comp_data_drosophila <- compcodeR::compData(drosophila, sample.annotations = coldata_compcodeR, info.pa
```

DESeq2 Analysis

To run the DESeq2 analysis we supplied a compcodeR data object containing a pseudo-count of our drosophila count matrix as the large number of zero values in the data obstructed the ability of DESeq2 to estimate size factors as a parameter. The runDiffExp function in compcodeR requires the path to the compcodeR data object, the name of the package, the name of the wrapper function of the packaage we would like to use and the ouput directory of the compcodeR results object.

```
comp_code_DEseq2 <- runDiffExp(data.file = "/datastore/2505621h/50cell_Analysis/comp_data_drosophila_ps
                               Rmdfunction = "DESeq2.createRmd",
                               output.directory = "/datastore/2505621h/50cell_Analysis/", fit.type = "p
                               test = "Wald", beta.prior = TRUE,
                               independent.filtering = TRUE, cooks.cutoff = TRUE,
                               impute.outliers = TRUE)
```

compcodeR allows you to use your own .rmd function for a differential expression analysis package of your choice that can be run with runDiffExp. However as fot this analysis we completed the differential expression outside of compcodeR we must complete some wrangling of our results tables for each package and coerce them into a compcodeR results object. The compcodeR results object requires:

- A method name - a list of a short name and the full name which will be used to label the method in the comparison plots
- The count matrix of the data
- The compcodeR sample annotation table we created at the beginning
- The parameters list
- The DE binary annotation of all genes
- The results matrix obtained from the DE package

The results tables may need a bit of wrangling and renaming to be formatted into the compcodeR results object, the most important aspects being:

- Columns containing p-values, adjusted p-values or FDR should be named pvalue, adjpvalue, and FDR respectively
- Row names should be the gene names **in the same order** as they appear in the count maxtrix
- Each result table must contain a column called score which is used by compcodeR to rank the genes in order of significance. This is calculated by 1-pvalue

Creating the compcodeR result objects

For each result object that is created ensure to save the .rds objects as we will need these for the final comparison.

```
# MAST

# Renaming & calculating score for compcodeR
fcHurdle$score <- 1-fcHurdle$`Pr(>Chisq)`
names(fcHurdle)[6] <- "FDR"
names(fcHurdle)[2] <- "pvalue"
fcHurdle <- fcHurdle[mixedorder(fcHurdle$primerid),]
fcHurdle <- tibble::column_to_rownames(fcHurdle, var = 'primerid')

# Creation of result object
```

```
method_name_MAST <- list(short.name = "MAST",
                         full.name = "MAST")

comp_data_drosophila_MAST <- compcodeR::compData(drosophila, sample.annotations = coldata_compcodeR, in

saveRDS(comp_data_drosophila_MAST, file = "/datastore/2505621h/50cell_Analysis/comp_data_drosophila_MAS
```

```
# DEsingle

# Formatting
dros_result_groups$score <- 1-dros_result_groups$pvalue.adj.FDR
dros_result_groups <- tibble::rownames_to_column(dros_result_groups, var = "gene")

dros_result_groups <- dros_result_groups[mixedorder(dros_result_groups$gene),]
rownames(dros_result_groups) <- NULL
dros_result_groups <- tibble::column_to_rownames(dros_result_groups, var = 'gene')
names(dros_result_groups)[21] <- "adjpvalue"

# Creation of result object
method_name_DEsingle <- list(short.name = "DEsingle",
                             full.name = "DEsingle")
comp_data_drosophila_DEsingle <- compcodeR::compData(drosophila, sample.annotations = coldata_compcodeR

saveRDS(comp_data_drosophila_DEsingle, file = "/datastore/2505621h/50cell_Analysis/comp_data_drosophila
```

```
#SigEMD

# Formatting
names(emd)[3] <- "adjpvalue"
emd$score <- 1-emd$adjpvalue

# Creation of object
method_name_SigEMD <- list(short.name = "SigEMD",
                           full.name = "SigEMD")

comp_data_drosophila_SigEMD <- compcodeR::compData(drosophila, sample.annotations = coldata_compcodeR,

saveRDS(comp_data_drosophila_SigEMD, file = "/datastore/2505621h/50cell_Analysis/comp_data_drosophila_S
```

```
# RankStat

# Formatting
result_RP$score <- 1-result_RP$p.value
result_DRP$score <- 1-result_DRP$p.value
result_RSum$score <- 1-result_RSum$p.value
result_RD$score <- 1-result_RD$p.value
result_R_RD$score <- 1-result_R_RD$p.value

# Creation of results objects

# RP
method_name_RP <- list(short.name = "RP",
                       full.name = "Rank Products")
```

```
comp_data_drosophila_RP <- compcodeR::compData(drosophila, sample.annotations = coldata_compcodeR, info

saveRDS(comp_data_drosophila_RP, file = "/datastore/2505621h/50cell_Analysis/comp_data_drosophila_RP.rds

# RSum
method_name_RSum <- list(short.name = "RS",
                         full.name = "Rank Sums")
comp_data_drosophila_RSum <- compcodeR::compData(drosophila, sample.annotations = coldata_compcodeR, in

saveRDS(comp_data_drosophila_RSum, file = "/datastore/2505621h/50cell_Analysis/comp_data_drosophila_RSu

# DRP
method_name_DRP <- list(short.name = "DRP",
                        full.name = "Difference of Rank Products")

comp_data_drosophila_DRP <- compcodeR::compData(drosophila, sample.annotations = coldata_compcodeR, inf

saveRDS(comp_data_drosophila_DRP, file = "/datastore/2505621h/50cell_Analysis/comp_data_drosophila_DRP.

# RD
method_name_RD <- list(short.name = "RD",
                       full.name = "Rank Distances")

comp_data_drosophila_RD <- compcodeR::compData(drosophila, sample.annotations = coldata_compcodeR, info

saveRDS(comp_data_drosophila_RD, file = "/datastore/2505621h/50cell_Analysis/comp_data_drosophila_RD.rds

# R_RD
method_name_R_RD <- list(short.name = "R_RD",
                         full.name = "Reverse Rank Distances")

comp_data_drosophila_R_RD <- compcodeR::compData(drosophila, sample.annotations = coldata_compcodeR, in

saveRDS(comp_data_drosophila_R_RD, file = "/datastore/2505621h/50cell_Analysis/comp_data_drosophila_R_R
```

Comparison of DE Results in compcodeR

Now that we have obtained all our `compcodeR` results objects we can proceed to the comparison. First we must create a `file.table` that is a dataframe containing the paths to all our results `.rds` files.

```
file.table <- data.frame(input.files = c("/datastore/2505621h/50cell_Analysis/comp_data_drosophila_pseu
                                         "/datastore/2505621h/50cell_Analysis/comp_data_drosophila_SigE
                                         "/datastore/2505621h/50cell_Analysis/comp_data_drosophila_MAST
                                         "/datastore/2505621h/50cell_Analysis/comp_data_drosophila_DEsi
                                         "/datastore/2505621h/50cell_Analysis/comp_data_drosophila_RP.r
                                         "/datastore/2505621h/50cell_Analysis/comp_data_drosophila_RSum
                                         "/datastore/2505621h/50cell_Analysis/comp_data_drosophila_DRP.
                                         "/datastore/2505621h/50cell_Analysis/comp_data_drosophila_RD.r
                                         "/datastore/2505621h/50cell_Analysis/comp_data_drosophila_R_RD
                         stringsAsFactors = FALSE)

parameters_comp <- list(incl.nbr.samples = NULL, incl.replicates = NULL,
```

```
                        incl.dataset = "Drosophila", incl.de.methods = NULL,
                        fdr.threshold = 0.05, tpr.threshold = 0.05,
                        typeI.threshold = 0.05, ma.threshold = 0.05,
                        fdc.maxvar = 1500, overlap.threshold = 0.05,
                        fracsign.threshold = 0.05,
                        comparisons = c("nbrtpfp", "overlap", "auc", "fdr", "tpr", "rocall", "correlati
```

We must also define the parameters for our comparison including the name of our dataset, specified thresholds, and the type of comparison we would like to do. For this analysis we specified to complete the following metrics:

- Number of true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN)
- Receiver Operating Characteristic (ROC) curve
- Area under the ROC curve
- False discovery rate (FDR)
- True positive rate (TPR)
- Sorensen index heatmap to compare similarity in returned DE genes between each package
- Spearmans correlation heatmap and dendrogram to compare the similarity in how the DE genes were ranked between each package

After these have been defined we can then run the comparison and define the ouput directory of where our results will be saved.

```
runComparison(file.table = file.table, parameters = parameters_comp, output.directory = "/datastore/250
```

The output of the analysis is a detailed webpage report that clearly explains each metric that was chosen in the comparison along with a plotted figure. Each figure in the `comcodeR` report is also saved in a separate folder as a `.pdf` and `.png` for use in other additional documents.

Calculation of additional summary statistics

A downside to completing the comparison analysis in `compcodeR` is that the package doesn't return any specific values for metrics such as the FDR for example despite the attractive plots it produces. Therefore these were calculated separately so that additional statistics could be caluclated such as the F1 score and precision of each package.

Firstly we need to use our groud truth list that contains the true DE genes and non-DE genes in our data

```
DE <- ground_truth[[1]]
nonDE <- ground_truth[[2]]
```

Using the MAST results as an example we can create a column that contains the names of all the genes in our results matrix by converting the row names. Then we subset the significant genes that have an FDR $<$ 0.05 into one dataframe and the other genes where the FDR $>=$ 0.05 are non-DE.

```
mast_results <- tibble::rownames_to_column(mast_results, var = "gene")
sig_mast_results <- subset(mast_results, mast_results$FDR < 0.05)
non_sig_mast_results <- subset(mast_results, mast_results$FDR >= 0.05)
```

We can then identify the four basic statistics as follows:

- TP = True Positives as being gene names that are in the significant results dataset and are also in the DE genes list
- FP = False Positives as being gene names that are in the significant results dataset but are also in the non-DE genes list
- TN = True Negatives as being the gene names in the non-significant results dataset that are also in the non-DE genes list
- FN = False Negatives as being the gene names that are in the non-significant results dataset and are also in the DE genes list

```
tp_mast <- sig_mast_results[sig_mast_results$gene %in% DE$`unlist(DE_genes)`,]

fp_mast <- sig_mast_results[sig_mast_results$gene %in% nonDE$`unlist(nonDE_genes)`,]

tn_mast <- non_sig_mast_results[non_sig_mast_results$gene %in% nonDE$`unlist(nonDE_genes)`,]

fn_mast <- non_sig_mast_results[non_sig_mast_results$gene %in% DE$`unlist(DE_genes)`,]
```

With these four statistics the length of each dataset was noted and entered into an Excel spreadsheet to calculate the FDR, TPR, F1 and precision. These can be calculated as follows:

- $$FDR = \frac{FP}{FP + TP}$$
- $$TPR = \frac{TP}{TP + FN}$$
- $$Precision = \frac{TP}{TP + FP}$$
- $$F1score = 2 * (\frac{Precision * Recall}{Precision + Recall})$$

A useful tutorial on how to calculate FDR and TPR in R is available here.

References

1. Baruzzo G, Patuzzi I, Di Camillo B. SPARSim single cell: a count data simulator for scRNA-seq data. Bioinformatics. 2020 Mar 1;36(5):1468–75.
2. Zappia L, Phipson B, Oshlack A. Splatter: simulation of single-cell RNA sequencing data. Genome Biol. 2017 Sep 12;18(1):174.
3. Li WV, Li JJ. A statistical simulator scDesign for rational scRNA-seq experimental design. Bioinformatics. 2019 Jul 15;35(14):i41–50.
4. Assefa AT, Vandesompele J, Thas O. SPsimSeq: semi-parametric simulation of bulk and single-cell RNA-sequencing data. Bioinformatics. 2020 May 1;36(10):3276–8.
5. Finak G, McDavid A, Yajima M, Deng J, Gersuk V, Shalek AK, et al. MAST: a flexible statistical framework for assessing transcriptional changes and characterizing heterogeneity in single-cell RNA sequencing data. Genome Biol. 2015 Dec 10;16(1):278.
6. Miao Z, Deng K, Wang X, Zhang X. DEsingle for detecting three types of differential expression in single-cell RNA-seq data. Bioinformatics. 2018 Sep 15;34(18):3223–4.

7. Wang T, Nabavi S. SigEMD: A powerful method for differential gene expression analysis in single-cell RNA sequencing data. Methods. 2018 Aug 1;145:25–32.

8. Breitling R, Armengaud P, Amtmann A, Herzyk P. Rank products: a simple, yet powerful, new method to detect differentially regulated genes in replicated microarray experiments. FEBS Lett. 2004 Aug 27;573(1–3):83–92.

9. Soneson C. compcodeR–an R package for benchmarking differential expression methods for RNA-seq data. Bioinforma Oxf Engl. 2014 Sep 1;30(17):2517–8.

10. Love MI, Huber W, Anders S. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. Genome Biol. 2014 Dec 5;15(12):550.