Differential state analysis with `muscat`

Helena L Crowell[1,2*]***, Charlotte Soneson***[1,3]***, Pierre-Luc Germain***[1,2] ***and Mark D Robinson***[1,2]

[1]Institute for Molecular Life Sciences, University of Zurich, Zurich, Switzerland
[2]Swiss Institute of Bioinformatics (SIB), Zurich, Switzerland
[3]Present address: Friedrich Miescher Institute Basel, Switzerland
& Swiss Institute of Bioinformatics (SIB), Basel, Switzerland

[*]helena.crowell@uzh.ch (mailto:helena.crowell@uzh.ch)

May 01, 2024

Abstract

Single-cell RNA sequencing (scRNA-seq) has quickly become an empowering technology to profile the transcriptomes of individual cells on a large scale. Many early analyses of differential expression have aimed at identifying differences between subpopulations, and thus are focused on finding subpopulation markers either in a single sample or across multiple samples. More generally, such methods can compare expression levels in multiple sets of cells, thus leading to cross-condition analyses.

However, given the emergence of replicated multi-condition scRNA-seq datasets, an area of increasing focus is making sample-level inferences, termed here as *differential state* (DS) analysis. For example, one could investigate the condition-specific responses of cell subpopulations measured from patients in each condition.

`muscat`: **mu**lti-sample **mu**lti-group **sc**RNA-seq **a**nalysis **t**ools (Crowell et al. 2020) provides various methods and visualization tools for DS analysis in multi-sample, multi-group, multi-(cell-)subpopulation scRNA-seq data, including cell-level mixed models and methods based on aggregated “pseudobulk” data, as well as a flexible simulation platform that mimics both single and multi-sample scRNA-seq data.

Package

muscat 1.19.0

Contents

For details on the concept and technicalities of DS analysis, and the methods presented here, consider having a look at our publication:

> Crowell HL, Soneson C*, Germain P-L*, Calini D, Collin L, Raposo C, Malhotra D, and Robinson MD: *muscat* detects subpopulation-specific state transitions from multi-sample multi-condition single-cell transcriptomics data.
> Nature Communications **11**, 6077 (2020).
> DOI: 10.1038/s41467-020-19894-4 (https://doi.org/10.1038/s41467-020-19894-4)

Load packages

```
library(dplyr)
library(ggplot2)
library(limma)
library(muscat)
library(purrr)
```

1 Introduction

1.1 What is DS analysis?

A fundamental task in the analysis of single-cell RNA-sequencing (scRNA-seq) data is the identification of systematic transcriptional changes (Stegle, Teichmann, and Marioni 2015). Such analyses are a critical step in the understanding of molecular responses, and have applications in development, in perturbation studies or in disease.
Most of the current scRNA-seq differential expression (DE) analysis methods are designed to test one set of cells against another (or more generally, multiple sets together), and can be used to compare cell clusters (e.g., for identifying marker genes) or across conditions (cells from one condition versus another) (Soneson and Robinson 2018). In such statistical models, the cells are the experimental units and thus represent the population that inferences will extrapolate to.

Using established terminology, we refer to cell *identity* as the combination of cell *type*, a stable molecular signature, and cell *state*, a transient snapshot of a cell's molecular events (Wagner, Regev, and Yosef 2016; Trapnell 2015). This classification is inherently arbitrary, but still provides a basis for biological interpretation and a framework for discovering interesting expression patterns from scRNA-seq datasets. For example, T cells could be defined as a single (albeit diverse) cell type or could be divided into discrete subtypes, if relevant information to categorize each cell at this level were available. In either case, the framework presented here would be able to focus on the cell type of interest and look for changes (in expression) across samples.
Given the emergence of multi-sample multi-group scRNA-seq datasets, the goal becomes making sample-level inferences (i.e., experimental units are samples). Thus, differential state (DS) analysis is defined as following a given cell type across a set of samples (e.g., individuals) and experimental conditions (e.g., treatments), in order to identify cell-type-specific responses, i.e., changes in cell state. DS analysis: i) should be able to detect diluted changes that only affect a single cell type, a subset of cell types or even a subset of a single subpopulation; and, ii) is intended to be orthogonal to clustering or cell type assignment.

1.2 Starting point

The starting point for a DS analysis is a (sparse) matrix of gene expression, either as counts or some kind of normalized data, where rows = genes and columns = cells. Each cell additionally has a cluster (subpopulation) label as well as a sample label; metadata should accompany the list of samples, such that they can be organized into comparable groups with sample-level replicates (e.g., via a design matrix).

The approach presented here is modular and thus subpopulation labels could originate from an earlier step in the analysis, such as clustering (Duò, Robinson, and Soneson 2018; Freytag et al. 2018), perhaps after integration (Butler et al. 2018; Stuart et al. 2019) or after labeling of clusters (Diaz-Mejia et al. 2019) or after cell-level type assignment (Zhang et al. 2019).

2 Getting started

2.1 Data description

For this vignette, we will use a *SingleCellExperiment (https://bioconductor.org/packages/3.20/SingleCellExperiment)* (SCE) containing 10x droplet-based scRNA-seq PBCM data from 8 Lupus patients obtained before and after 6h-treatment with IFN-β (Kang et al. 2018). The complete raw data, as well as gene and cell metadata is available through the NCBI GEO, accession number GSE96583 (https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE96583).

2.2 Loading the data

The Kang et al. (2018) dataset has been made available through Bioconductor's `ExperimentHub` and can be loaded into R as follows: We first initialize a Hub instance to search for and load available data with the `ExperimentHub` function, and store the complete list of records in the variable `eh`. Using `query`, we then retrieve any records that match our keyword(s) of interest, as well as their corresponding accession ID (EH1234).

```
library(ExperimentHub)
eh <- ExperimentHub()
query(eh, "Kang")
```

```
## ExperimentHub with 3 records
## # snapshotDate(): 2024-04-29
## # $dataprovider: NCI_GDC, GEO
## # $species: Homo sapiens
## # $rdataclass: character, SingleCellExperiment, BSseq
## # additional mcols(): taxonomyid, genome, description,
## #   coordinate_1_based, maintainer, rdatadateadded, preparercla
## #   rdatapath, sourceurl, sourcetype
## # retrieve records with, e.g., 'object[["EH1661"]]'
##
##            title
##   EH1661 | Whole Genome Bisulfit Sequencing Data for 47 samples
##   EH1662 | Whole Genome Bisulfit Sequencing Data for 47 samples
##   EH2259 | Kang18_8vs8
```

Finally, we load the data of interest into R via `[[` and the corresponding accession ID. The dataset contains >35,000 genes and ~29,000 cells:

```
(sce <- eh[["EH2259"]])
```

```
## class: SingleCellExperiment
## dim: 35635 29065
## metadata(0):
## assays(1): counts
## rownames(35635): MIR1302-10 FAM138A ... MT-ND6 MT-CYB
## rowData names(2): ENSEMBL SYMBOL
## colnames(29065): AAACATACAATGCC-1 AAACATACATTTCC-1 ... TTTGCATG
##   TTTGCATGTCTTAC-1
## colData names(5): ind stim cluster cell multiplets
## reducedDimNames(1): TSNE
## mainExpName: NULL
## altExpNames(0):
```

2.3 Preprocessing

The *scater (https://bioconductor.org/packages/3.20/scater)* package (McCarthy et al. 2017) provides a variety of tools for preprocessing and quality control of single-cell transcriptomic data. For completeness, we will apply some minimal filtering steps to

- remove undetected genes
- remove cells with very few or many detected genes
- remove very lowly expressed genes
- compute normalized expression values for visualization

For more thorough preprocessing, we refer to the Quality control with *scater* (https://bioconductor.org/packages/release/bioc/vignettes/scater/inst/doc/vignette-qc.html) vignette.

```
# remove undetected genes
sce <- sce[rowSums(counts(sce) > 0) > 0, ]
dim(sce)
```

```
## [1] 18890 29065
```

We use `perCellQCMetrics` to compute various per-cell quality control metrics, and proceed with filtering cells and genes as noted above:

```
# calculate per-cell quality control (QC) metrics
library(scater)
qc <- perCellQCMetrics(sce)

# remove cells with few or many detected genes
ol <- isOutlier(metric = qc$detected, nmads = 2, log = TRUE)
sce <- sce[, !ol]
dim(sce)
```

```
## [1] 18890 26820
```

```
# remove lowly expressed genes
sce <- sce[rowSums(counts(sce) > 1) >= 10, ]
dim(sce)
```

```
## [1]  7118 26820
```

Finally, we use `logNormCounts` to calculate $\log_2$-transformed normalized expression values by dividing each count by its size factor, adding a pseudo-count of 1, and log-transforming[1].

[1] Note that, in this workflow, expression values are used for visualization only, and that differential analyses are performed on pseudobulks (section 3.3) or the count data directly (section 3.4).

```
# compute sum-factors & normalize
sce <- computeLibraryFactors(sce)
sce <- logNormCounts(sce)
```

Alternatively, expression values could be obtained via `vst` (variance stabilizing transformation) from the *sctransform (https://CRAN.R-project.org/package=sctransform)* package (Hafemeister and Satija 2019), which returns Pearson residuals from a regularized negative binomial regression model that can be interpreted as normalized expression values:

```
library(sctransform)
assays(sce)$vstresiduals <- vst(counts(sce), verbosity = FALSE)$y
```

By default, *scater (https://bioconductor.org/packages/3.20/scater)*'s functions will try to access the assay data specified via argument `exprs_values` (default `logcounts`) for e.g. visualization and dimension reduction. When an alternative assay such as the `vstresiduals` above should be used, it is thus necessary to explicitly specify this, for example, via `runUMAP(sce, exprs_values = "vstresiduals")` to compute UMAP cell embeddings on the assay data compute above.

2.4 Data preparation

`muscat` expects a certain format of the input SCE. Specifically, the following cell metadata (`colData`) columns have to be provided:

- `"sample_id"`: unique sample identifiers (e.g., PeterPan_ref1, Nautilus_trt3, …)
- `"cluster_id"`: subpopulation (cluster) assignments (e.g., T cells, monocytes, …)
- `"group_id"`: experimental group/condition (e.g., control/treatment, healthy/diseased, …)

```
sce$id <- paste0(sce$stim, sce$ind)
(sce <- prepSCE(sce,
    kid = "cell", # subpopulation assignments
    gid = "stim",  # group IDs (ctrl/stim)
    sid = "id",   # sample IDs (ctrl/stim.1234)
    drop = TRUE))  # drop all other colData columns
```

```
## class: SingleCellExperiment
## dim: 7118 26820
## metadata(1): experiment_info
## assays(2): counts logcounts
## rownames(7118): NOC2L HES4 ... S100B PRMT2
## rowData names(2): ENSEMBL SYMBOL
## colnames(26820): AAACATACAATGCC-1 AAACATACATTTCC-1 ... TTTGCATG
##   TTTGCATGTCTTAC-1
## colData names(3): cluster_id sample_id group_id
## reducedDimNames(1): TSNE
## mainExpName: NULL
## altExpNames(0):
```

For consistency and easy accession throughout this vignette, we will store cluster and sample IDs, as well as the number of clusters and samples into the following *simple* variables:

```
nk <- length(kids <- levels(sce$cluster_id))
ns <- length(sids <- levels(sce$sample_id))
names(kids) <- kids; names(sids) <- sids
```

2.5 Data overview

2.5.1 Cluster-sample sizes

As we will be aggregating measurements at the cluster-sample level, it is of particular importance to check the number of cells captured for each such instance. While `aggregateData` (see Section 3.1) allows excluding cluster-sample combinations with less than a threshold number of cells, clusters or samples with overall very low cell-counts may be excluded from further analysis at this point already.

For the Kang et al. (2018) dataset, for example, one might consider removing the *Dendritic cells* and *Megakaryocytes* clusters, as these contain less than 50 cells across all samples.

```
# nb. of cells per cluster-sample
t(table(sce$cluster_id, sce$sample_id))
```

```
##
##           B cells CD14+ Monocytes CD4 T cells CD8 T cells Denc
##   ctrl101      113             186         336          95
##   ctrl1015     476             783         919         226
##   ctrl1016     144             419         526         671
##   ctrl1039      30             116         202          30
##   ctrl107       51             222         197          31
##   ctrl1244     134             429        1215          82
##   ctrl1256     240             383        1136         156
##   ctrl1488     234             317        1343          78
##   stim101      144             222         437         121
##   stim1015     357             683         814         153
##   stim1016     129             361         426         600
##   stim1039      39             154         318          40
##   stim107       56             185         217          22
##   stim1244      94             318         980          46
##   stim1256     211             369        1047         133
##   stim1488     283             370        1658          73
##
##           FCGR3A+ Monocytes Megakaryocytes NK cells
##   ctrl101                81             12       84
##   ctrl1015              232             25      208
##   ctrl1016              126             15      151
##   ctrl1039               28              5       20
##   ctrl107                29              5       49
##   ctrl1244               53             19      131
##   ctrl1256               50             15      275
##   ctrl1488               99             21      120
##   stim101               126              7      120
##   stim1015              222             24      224
##   stim1016              124             12      239
##   stim1039               36             13       32
##   stim107                36              4       51
##   stim1244               35             14      136
##   stim1256               73             21      257
##   stim1488              139             35      187
```

2.5.2 Dimension reduction

The dimension reductions (DR) available within the SCE can be accessed via `reducedDims` from the *scater (https://bioconductor.org/packages/3.20/scater)* package. The data provided by Kang et al. (2018) already contains t-SNE coordinates; however, we can of course compute additional dimension reductions using one of *scater (https://bioconductor.org/packages/3.20/scater)*'s `runX` functions:

```
# compute UMAP using 1st 20 PCs
sce <- runUMAP(sce, pca = 20)
```

Using *scater (https://bioconductor.org/packages/3.20/scater)*'s `plotReducedDim` function, we can plot t-SNE and UMAP representations colored by cluster and group IDs, respectively. We additionally create a small wrapper function, `.plot_dr()`, to improve the readability of color legends and simplify the plotting theme:

```
# wrapper to prettify reduced dimension plots
.plot_dr <- function(sce, dr, col)
  plotReducedDim(sce, dimred = dr, colour_by = col) +
    guides(fill = guide_legend(override.aes = list(alpha = 1, size
    theme_minimal() + theme(aspect.ratio = 1)
```

For our dataset, the t-SNE and UMAP colored by `cluster_id` s show that cell-populations are well-separated from one another. IFN-β stimulation manifests as a severe shift in the low-dimensional projection of cells when coloring by `group_id` s, indicating widespread, genome-scale transcriptional changes.

```
# downsample to max. 100 cells per cluster
cs_by_k <- split(colnames(sce), sce$cluster_id)
cs100 <- unlist(sapply(cs_by_k, function(u)
  sample(u, min(length(u), 100))))

# plot t-SNE & UMAP colored by cluster & group ID
for (dr in c("TSNE", "UMAP"))
  for (col in c("cluster_id", "group_id"))
    .plot_dr(sce[, cs100], dr, col)
```

TSNE | UMAP

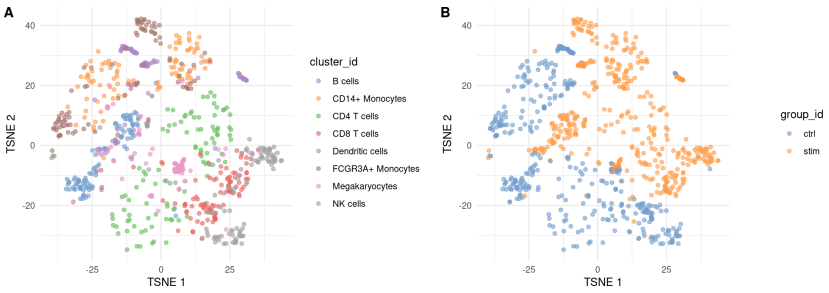


Figure 1: **Dimension reduction plots**
Cells are colored by cluster ID (A) and group ID (B), respectively. For each cluster, at most 100 cells were sampled for plotting.

3 Differential State (DS) analysis

To test for state changes across conditions, we will consider two types of approaches: i) mixed models that act directly on cell-level measurements; and ii) aggregation-based methods that act on *pseudobulk* data. For both approaches, each gene is tested for state changes in each cluster. Thus, a total of $\#(genes) \times \#(clusters)$ tests will be performed per comparison of interest. The following schematic summarizes the data representation considered by cell- and sample-level approaches, respectively:

Figure 3: **Schematic overview of cell- and sample-level approaches for DS analysis**
Top panels show a schematic of the data distributions or aggregates across samples (each violin is a group or sample; each dot is a sample) and conditions (blue or orange). The bottom panels highlight the data organization in sub-matrix slices of the original count table.

3.1 Aggregation of single-cell to pseudobulk data

In order to leverage existing robust bulk RNA-seq DE frameworks, such as *edgeR (https://bioconductor.org/packages/3.20/edgeR)* (Robinson, McCarthy, and Smyth 2010), *DESeq2 (https://bioconductor.org/packages/3.20/DESeq2)* (Love, Huber, and Anders 2014), and *limma (https://bioconductor.org/packages/3.20/limma)* (Ritchie et al. 2015), we first aggregate measurements for each sample (in each cluster) to obtain pseudobulk data.

In general, `aggregateData()` will aggregate the data by the `colData` variables specified with argument `by`, and return a `SingleCellExperiment` containing pseudobulk data.
For DS analysis, measurements must be aggregated at the cluster-sample level (default `by = c("cluster_id", "sample_id")`). In this case, the returned `SingleCellExperiment` will contain one assay per cluster, where rows = genes and columns = samples. Arguments `assay` and `fun` specify the input data and summary statistic, respectively, to use for aggregation.
While, in principle, various combinations of input data (raw/(log-)normalized counts, CPM ect.) and summary statistics (sum, mean, median) could be applied, we here default to the sum of raw counts:

```
pb <- aggregateData(sce,
    assay = "counts", fun = "sum",
    by = c("cluster_id", "sample_id"))
# one sheet per subpopulation
assayNames(pb)
```

```
## [1] "B cells"           "CD14+ Monocytes"   "CD4 T cells"
## [4] "CD8 T cells"       "Dendritic cells"   "FCGR3A+ Monocytes'
## [7] "Megakaryocytes"    "NK cells"
```

```
# pseudobulks for 1st subpopulation
t(head(assay(pb)))
```

```
##          NOC2L HES4 ISG15 TNFRSF18 TNFRSF4 SDF4
## ctrl101     12    0    13        8       1   13
## ctrl1015    37    4   136       55      16   42
## ctrl1016    13    3    42       10       0   15
## ctrl1039     2    1    16        3       1    1
## ctrl107      7    0     6        3       4    5
## ctrl1244    24    7    25       30      11    7
## ctrl1256    22    1    65       30      11   17
## ctrl1488    19    1    34       27       6   23
## stim101     12    8  1362        3       1    9
## stim1015    34   27  4022       21       2   19
## stim1016     7    8  1271        2       4    6
## stim1039     3    3   393        1       1    2
## stim107      4    2   556        1       0    3
## stim1244    13   10   830        8       2    5
## stim1256    14   10  2235       10       2    6
## stim1488    18    5  2927       13       1   19
```

3.2 Pseudobulk-level MDS plot

Prior to conducting any formal testing, we can compute a multi-dimensional scaling (MDS) plot of aggregated signal to explore overall sample similarities.

`pbMDS` takes as input any SCE containg PB data as returned by `aggregateData`, and computes MDS dimensions using *edgeR (https://bioconductor.org/packages/3.20/edgeR)*. Ideally, such a representation of the data should separate both clusters and groups from one another. Vice versa, samples from the same cluster or group should cluster together.

In our MDS plot on pseudo-bulk counts (Fig. 4), we can observe that the first dimension (MDS1) clearly separates cell populations (clusters), while the second (MDS2) separates control and stimulated samples (groups). Furthermore, the two T-cell clusters fall close to each other.

```
(pb_mds <- pbMDS(pb))
```

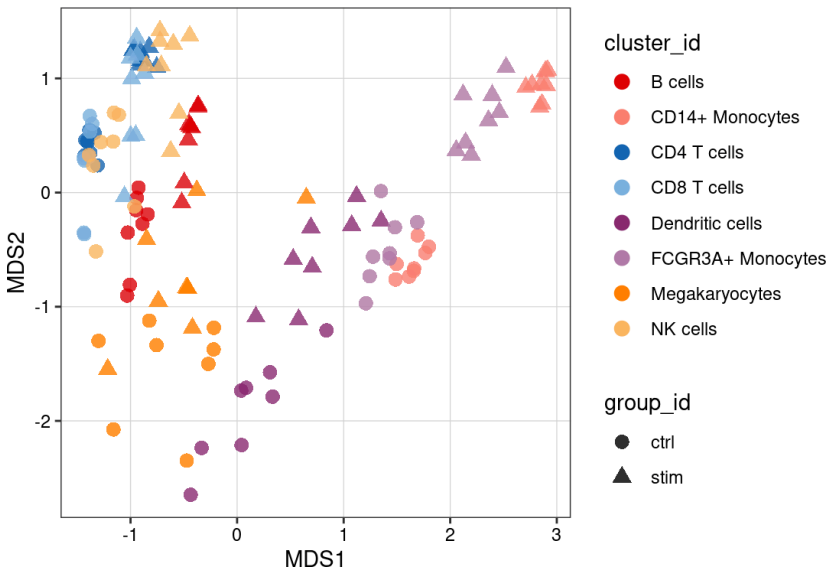


Figure 4: **Pseudobulk-level multidimensional scaling (MDS) plot**
Each point represents a cluster-sample instance; points are colored by cluster ID and shaped by group ID.

If you're not satisfied with how the plot looks, here's an example of how to modify the `ggplot`-object from above in various ways:

```
# use very distinctive shaping of groups & change cluster colors
pb_mds <- pb_mds +
  scale_shape_manual(values = c(17, 4)) +
  scale_color_manual(values = RColorBrewer::brewer.pal(8, "Set2"))
# change point size & alpha
pb_mds$layers[[1]]$aes_params$size <- 5
pb_mds$layers[[1]]$aes_params$alpha <- 0.6
pb_mds
```

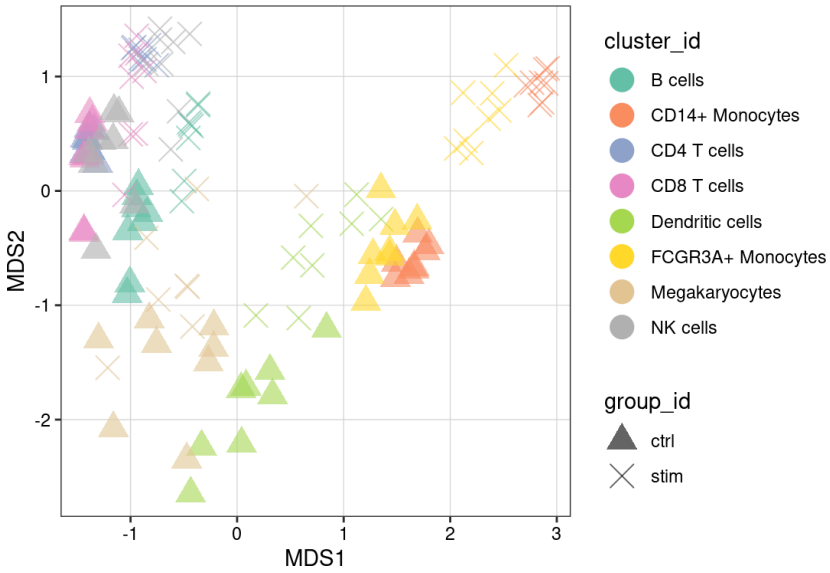


Figure 5: **Pseudobulk-level MDS plot v2**
Default plotting aesthetics were modified to change shaping of groups, coloring of clusters, as well as point size and transparency.

3.3 Sample-level analysis: Pseudobulk methods

Once we have assembled the pseudobulk data, we can test for DS using `pbDS`. By default, a $\sim group\_id$ model is fit, and the last coefficient of the linear model is tested to be equal to zero.

```
# run DS analysis
res <- pbDS(pb, verbose = FALSE)
# access results table for 1st comparison
tbl <- res$table[[1]]
# one data.frame per cluster
names(tbl)
```

```
## [1] "B cells"           "CD14+ Monocytes"   "CD4 T cells"
## [4] "CD8 T cells"       "Dendritic cells"   "FCGR3A+ Monocytes'
## [7] "Megakaryocytes"    "NK cells"
```

```
# view results for 1st cluster
k1 <- tbl[[1]]
head(format(k1[, -ncol(k1)], digits = 2))
```

```
##       gene cluster_id    logFC logCPM       F   p_val p_adj.loc
## 1    NOC2L    B cells -0.29137    6.5 2.0e+00 1.7e-01   3.2e-01
## 2    ISG15    B cells  5.47364   12.2 9.8e+02 1.0e-21   3.8e-19
## 3 TNFRSF18    B cells -1.35153    6.4 2.6e+01 2.8e-05   1.9e-04
## 4   CPSF3L    B cells -0.14759    6.1 4.1e-01 5.3e-01   6.8e-01
## 5 AURKAIP1    B cells -0.01160    7.8 8.6e-03 9.3e-01   9.6e-01
## 6   MRPL20    B cells  0.23488    7.4 2.5e+00 1.3e-01   2.5e-01
```

Depening on the complexity of the experimental design (e.g., when there are more than two groups present), comparison(s) of interest may need to be specified explicitly. We can provide `pbDS` with a design matrix capturing the experimental design using `model.matrix` (package *stats*), and a contrast matrix that specifies our comparison of interesting using `makeContrasts` from

the *limma (https://bioconductor.org/packages/3.20/limma)* package. Alternatively, the comparison(s) of interest (or a list thereof) can be specified with via `coefs` (see `?glmQLFTest` for details). For the Kang et al. (2018) dataset, we want to carry out a single comparison of stimulated against control samples, thus placing `"ctrl"` on the right-hand side as the reference condition:

```
# construct design & contrast matrix
ei <- metadata(sce)$experiment_info
mm <- model.matrix(~ 0 + ei$group_id)
dimnames(mm) <- list(ei$sample_id, levels(ei$group_id))
contrast <- makeContrasts("stim-ctrl", levels = mm)

# run DS analysis
pbDS(pb, design = mm, contrast = contrast)
```

3.4 Cell-level analysis: Mixed models

Alternative to the above sample-level approach, we fit (for each gene) a mixed model (MM) to the cell-level measurement data. `muscat` provides implementations of MM that use 3 main approaches:

1. fitting linear mixed models (LMMs) on log-normalized data with observational weights,
2. fitting LMMs on variance-stabilized data; and,
3. fitting generalized linear mixed models (GLMMs) directly on counts

In each case, a $\sim 1 + \text{group\_id} + (1 \mid \text{sample\_id})$ model is fit for each gene, optimizing the log-likelihood (i.e., `REML = FALSE`). P-values are calculated using the estimates of degrees of freedom specifying by argument `df` (default `"Satterthwaite"`). Fitting, testing and moderation are applied subpopulation-wise. For differential testing, `mmDS` will only consider:

- subpopulations with at least `n_cells` cells (default 10) in at least `n_samples` samples (default 2)
- genes with a count >= `min_count` (default 1) in at least `min_cells` (default 20)

Mixed model based approaches can be run directly on cell-level measurements, and do not require prior aggregation:

```
# 1st approach
mm <- mmDS(sce, method = "dream",
  n_cells = 10, n_samples = 2,
  min_counts = 1, min_cells = 20)

# 2nd & 3rd approach
mm <- mmDS(sce, method = "vst", vst = "sctransform")
mm <- mmDS(sce, method = "nbinom")
```

4 Handling results

4.1 Results filtering & overview

To get a general overview of the differential testing results, we first filter them to retain hits FDR < 5% and abs(logFC) > 1, and count the number and frequency of differential findings by cluster. Finally, we can view the top hits (lowest adj. p-value) in each cluster.

```
# filter FDR < 5%, abs(logFC) > 1 & sort by adj. p-value
tbl_fil <- lapply(tbl, function(u) {
  u <- dplyr::filter(u, p_adj.loc < 0.05, abs(logFC) > 1)
  dplyr::arrange(u, p_adj.loc)
})

# nb. of DS genes & % of total by cluster
n_de <- vapply(tbl_fil, nrow, numeric(1))
p_de <- format(n_de / nrow(sce) * 100, digits = 3)
data.frame("#DS" = n_de, "%DS" = p_de, check.names = FALSE)
```

```
##                     #DS    %DS
## B cells            238  3.344
## CD14+ Monocytes   1049 14.737
## CD4 T cells        323  4.538
## CD8 T cells         93  1.307
## Dendritic cells    117  1.644
## FCGR3A+ Monocytes  386  5.423
## Megakaryocytes      28  0.393
## NK cells           171  2.402
```

```
# view top 2 hits in each cluster
top2 <- bind_rows(lapply(tbl_fil, top_n, 2, p_adj.loc))
format(top2[, -ncol(top2)], digits = 2)
```

```
##        gene        cluster_id logFC logCPM    F   p_val p_adj.l
## 1   DNAJC15           B cells   1.1    7.1 12.6 0.00155    0.00
## 2      MCM5           B cells  -1.1    6.4 10.1 0.00387    0.01
## 3     ASGR1   CD14+ Monocytes  -1.2    5.2  9.2 0.00650    0.01
## 4  SERPINB2   CD14+ Monocytes  -1.1    7.2  7.9 0.01053    0.01
## 5      HOPX       CD4 T cells  -1.0    4.1  8.2 0.00846    0.04
## 6     CPNE2       CD4 T cells  -1.1    4.1  8.2 0.00858    0.04
## 7    TUBA4A       CD8 T cells  -1.1    7.3 17.5 0.00033    0.00
## 8      XBP1       CD8 T cells   1.0    8.3 15.9 0.00055    0.00
## 9      RALA   Dendritic cells  -1.0    9.1  8.6 0.01030    0.03
## 10    H2AFZ   Dendritic cells  -1.0   10.2  7.3 0.01599    0.04
## 11  DNAJC15 FCGR3A+ Monocytes   1.1    7.1  7.9 0.01025    0.02
## 12    CBWD1 FCGR3A+ Monocytes   1.1    6.6  7.5 0.01236    0.02
## 13     SAT1    Megakaryocytes   1.4   12.0 19.0 0.00056    0.00
## 14  TMEM123    Megakaryocytes   1.1    9.7 19.2 0.00055    0.00
## 15   NT5C3A    Megakaryocytes   1.3    9.7 14.6 0.00169    0.01
## 16    FOXP1          NK cells  -1.0    6.7 10.7 0.00305    0.01
## 17    ITM2C          NK cells  -1.1    7.4 10.7 0.00306    0.01
```

4.2 Calculating expression frequencies

Besides filter DS results based on magnitude (logFCs) and significance (FDR), it is often worthwhile to also consider the expression frequencies of each gene, i.e., the fraction of cells that express a given gene in each sample and/or group.

`muscat` provides wrapper, `calcExprFreqs` to compute cluster-sample/-group wise expression frequencies. Here, a gene is considered to be expressed when the specified measurement value (argument `assay`) falls above a certain threshold (argument `th`). Note that, `assay = "counts"` and `th = 0` (default) amounts to the fraction of cells for which a respective gene has been detected.

`calcExprFreqs` will return a *SingleCellExperiment (https://bioconductor.org/packages/3.20/SingleCellExperiment)* object, where sheets (assays) = clusters, rows = genes, and columns = samples (and groups, if `group_id`s are present in the `colData` of the input SCE).

```
frq <- calcExprFreqs(sce, assay = "counts", th = 0)
# one sheet per cluster
assayNames(frq)
```

```
## [1] "B cells"           "CD14+ Monocytes"   "CD4 T cells"
## [4] "CD8 T cells"       "Dendritic cells"   "FCGR3A+ Monocytes'
## [7] "Megakaryocytes"    "NK cells"
```

```
# expression frequencies in each
# sample & group; 1st cluster
t(head(assay(frq), 5))
```

```
##               NOC2L        HES4      ISG15    TNFRSF18     TNFR
## ctrl101  0.09734513 0.000000000 0.08849558 0.061946903 0.008849
## ctrl1015 0.07773109 0.008403361 0.18067227 0.090336134 0.021008
## ctrl1016 0.08333333 0.013888889 0.20833333 0.048611111 0.000000
## ctrl1039 0.06666667 0.033333333 0.33333333 0.100000000 0.033333
## ctrl107  0.09803922 0.000000000 0.07843137 0.058823529 0.019607
## ctrl1244 0.12686567 0.037313433 0.10447761 0.126865672 0.029850
## ctrl1256 0.08750000 0.004166667 0.18750000 0.091666667 0.025000
## ctrl1488 0.06837607 0.004273504 0.11965812 0.076923077 0.012820
## stim101  0.08333333 0.055555556 0.98611111 0.020833333 0.006944
## stim1015 0.08963585 0.072829132 0.99719888 0.044817927 0.005602
## stim1016 0.03875969 0.054263566 1.00000000 0.007751938 0.023255
## stim1039 0.05128205 0.051282051 1.00000000 0.025641026 0.025641
## stim107  0.05357143 0.035714286 1.00000000 0.017857143 0.000000
## stim1244 0.11702128 0.095744681 0.98936170 0.053191489 0.021276
## stim1256 0.06635071 0.037914692 0.99526066 0.037914692 0.004739
## stim1488 0.05653710 0.014134276 0.99646643 0.031802120 0.003533
## ctrl     0.08509142 0.009845288 0.15963432 0.084388186 0.018284
## stim     0.07235339 0.050266565 0.99543031 0.033511043 0.008377
```

We can use the obtained frequencies to, for instance, only retain genes that are expressed in an average of 10% of cells in at least 1 group:

```
gids <- levels(sce$group_id)
frq10 <- vapply(as.list(assays(frq)),
  function(u) apply(u[, gids] > 0.1, 1, any),
  logical(nrow(sce)))
t(head(frq10))
```

```
##                   NOC2L  HES4 ISG15 TNFRSF18 TNFRSF4  SDF4
## B cells           FALSE FALSE  TRUE    FALSE   FALSE FALSE
## CD14+ Monocytes   FALSE  TRUE  TRUE    FALSE   FALSE  TRUE
## CD4 T cells       FALSE FALSE  TRUE    FALSE   FALSE FALSE
## CD8 T cells       FALSE FALSE  TRUE    FALSE   FALSE FALSE
## Dendritic cells   FALSE  TRUE  TRUE    FALSE    TRUE  TRUE
## FCGR3A+ Monocytes FALSE  TRUE  TRUE    FALSE   FALSE FALSE
## Megakaryocytes    FALSE FALSE  TRUE    FALSE   FALSE FALSE
## NK cells          FALSE FALSE  TRUE     TRUE   FALSE  TRUE
```

```
tbl_fil2 <- lapply(kids, function(k)
  dplyr::filter(tbl_fil[[k]],
    gene %in% names(which(frq10[, k]))))

# nb. of DS genes & % of total by cluster
n_de <- vapply(tbl_fil2, nrow, numeric(1))
p_de <- format(n_de / nrow(sce) * 100, digits = 3)
data.frame("#DS" = n_de, "%DS" = p_de, check.names = FALSE)
```

```
##                   #DS   %DS
## B cells           227 3.189
## CD14+ Monocytes   623 8.752
## CD4 T cells       155 2.178
## CD8 T cells        93 1.307
## Dendritic cells   117 1.644
## FCGR3A+ Monocytes 384 5.395
## Megakaryocytes     28 0.393
## NK cells          163 2.290
```

4.3 Formatting results

Especially when testing multiple contrasts or coefficients, the results returned by `runDS` may become very complex and unhandy for exploration or exporting. Results can be formatted using `resDS`, which provides two alternative modes for formatting: `bind = "row"/"col"`.

When `bind = "row"`, results from all comparisons will be merged vertically (analogous to `do.call("rbind", ...)`) into a tidy format table, with column `contrast/coef` specifying the comparison.

Otherwise, `bind = "col"`, results will be merged horizontally into a single wide table where all results for a given gene and cluster are kept in one row. An identifier of the respective contrast of coefficient is then appended to the column names. This format is useful when wanting to view a specific gene's behavior across, for example, multiple treatments, but will become *messy* when many comparisons are included.

Expression frequencies computed with `calcExprFreqs`, as well as cluster-sample level avg. CPM, can be included in the results by setting `frq/cpm = TRUE`. Alternatively, if the former have been pre-computed, they can be supplied directly as an input to `resDS` (see example below).

```
# tidy format; attach pre-computed expression frequencies
resDS(sce, res, bind = "row", frq = frq)

# big-table (wide) format; attach CPMs
resDS(sce, res, bind = "col", cpm = TRUE)
```

Alternatively, if expression frequencies have not been pre-computed with `calcExprFreqs`, they may be added to the results table directly by specifying `frq = TRUE`:

```
# compute expression frequencies on the fly
resDS(sce, res, frq = TRUE)
```

5 Visualizing results

5.1 Between-cluster concordance

DS analysis aims at identifying population-specific changes in state (or expression) across conditions. In this setting, key questions of interest arise, e.g., which genes are DE in only a single (or very few) clusters? How many DE genes are shared between clusters? In summary, what is the general concordance in differential findings between clusters?

To gain an impression of the between-cluster (dis-)agreement on DE genes, we generate an UpSet-plot that visualizes the number of DE genes that are shared across or unique to certain clusters:

```
library(UpSetR)
de_gs_by_k <- map(tbl_fil, "gene")
upset(fromList(de_gs_by_k))
```

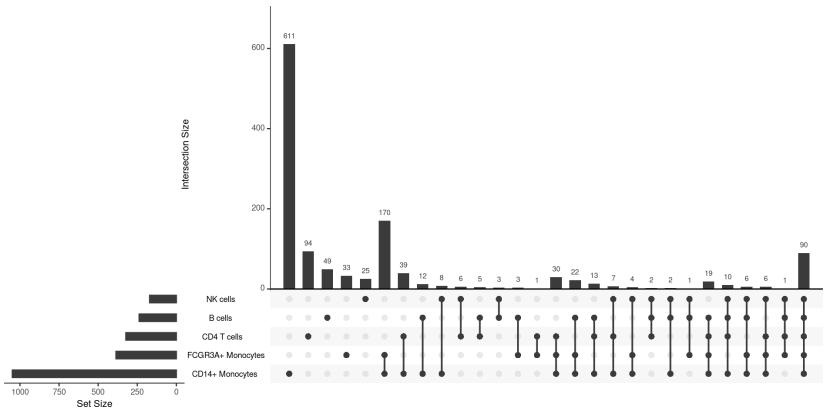


Figure 6: **Upset plot**
Included are DS findings (FDR < 0.05, |logFC| > 1) across all clusters; shown are the 50 most frequent interactions.

An UpSet plot as the one above tells us, for instance, that 185 genes are differential for all subpopulations; 387 across both *Monocytes* clusters; and 159 only in the *B cells* cluster.

5.2 DR colored by expression

The code chunk generates a set of t-SNEs colored by gene expression for the top-8 DS genes. To match the affected cells to their cluster and experimental group, see the t-SNEs colored by cluster and group ID from above.

```
# pull top-8 DS genes across all clusters
top8 <- bind_rows(tbl_fil) %>%
  slice_min(p_adj.loc, n = 8,
    with_ties = FALSE) %>%
  pull("gene")

# for ea. gene in 'top8', plot t-SNE colored by its expression
ps <- lapply(top8, function(g)
  .plot_dr(sce[, cs100], "TSNE", g) +
    ggtitle(g) + theme(legend.position = "none"))

# arrange plots
plot_grid(plotlist = ps, ncol = 4, align = "vh")
```

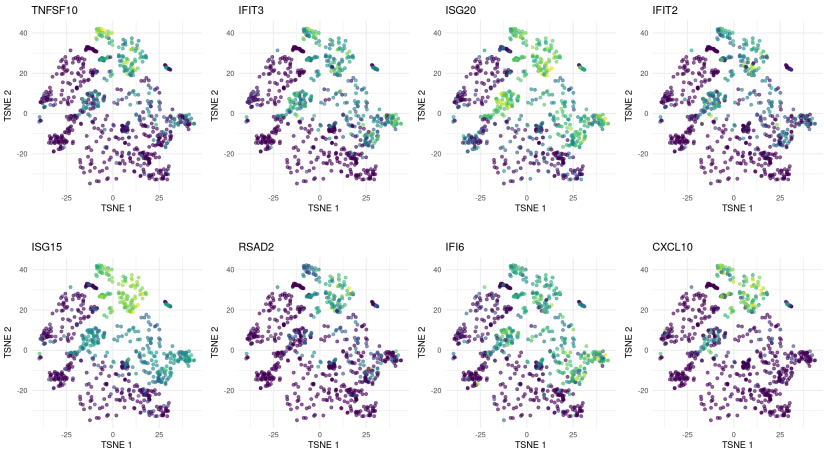


Figure 7: **t-SNE colored by gene expression**
Show are t-SNE projections with cells colored by the expression of the top-8 DS genes. For each cluster, at most 100 cells were sampled for plotting.

5.3 Cell-level viz.: Violin plots

For changes of high interest, we can view the cell-level expression profiles of a specific gene across samples or groups using `plotExpression` (*scater (https://bioconductor.org/packages/3.20/scater)* package). Here, we generate violin plots for the top-6 DS genes (lowest adj. p-value) in the *B cells* cluster[2].

[2] Note that, as DS testing is done at the cluster-level, we need to subset the cells that have been assigned to the corresponding cluster for plotting.

```
plotExpression(sce[, sce$cluster_id == "B cells"],
  features = tbl_fil$`B cells`$gene[seq_len(6)],
  x = "sample_id", colour_by = "group_id", ncol = 3) +
  guides(fill = guide_legend(override.aes = list(size = 5, alpha =
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

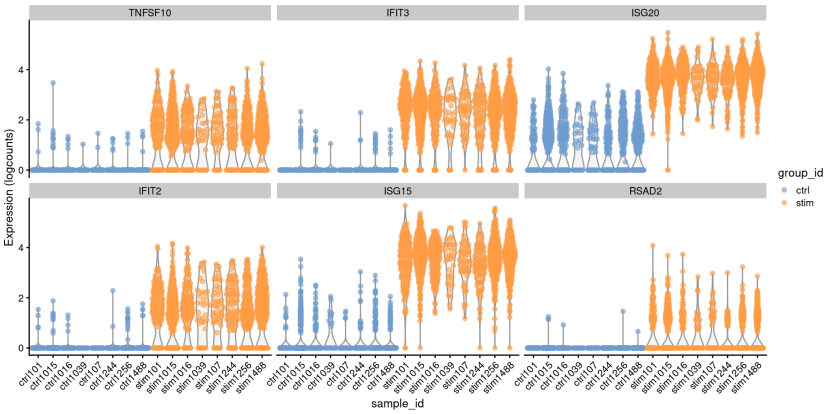
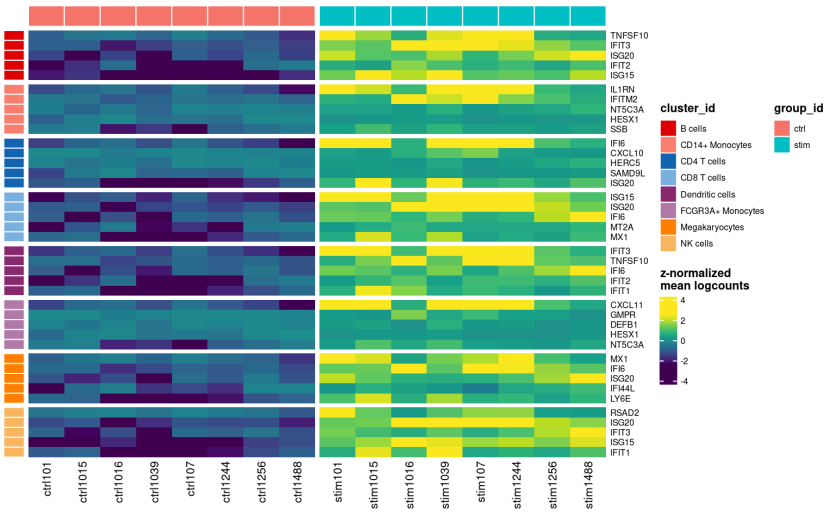


Figure 8: **Violin plots**
Show are the top 6 hits (lowest adj. p-value) for the B cells cluster. Each violin is a sample; points are colored by group ID.

5.4 Sample-level viz.: Pseudobulk heatmaps

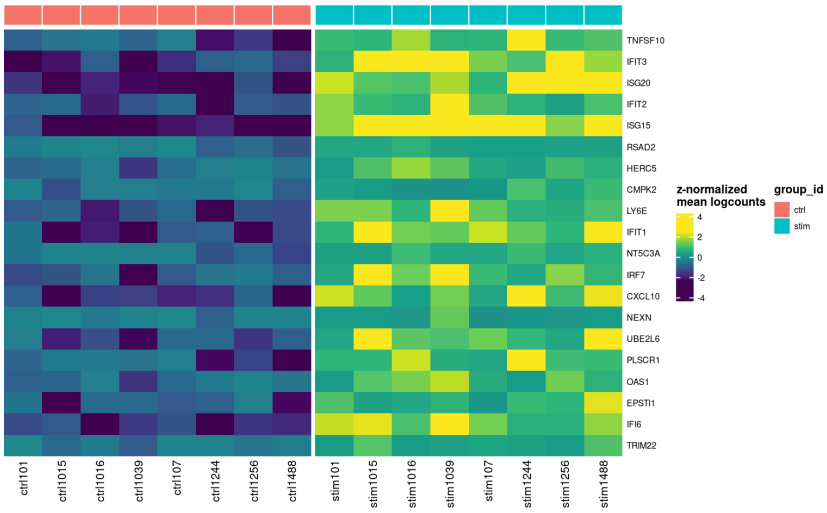
Especially when wanting to gain an overview of numerous DE testing results for many clusters, both dimension reduction and cell-level visualizations require a lot of space can become cumbersome to interpret. In this setting, it is thus recommended to visualize aggregated measures, e.g., mean expressions by cluster sample.

```
# top-5 DS genes per cluster
pbHeatmap(sce, res, top_n = 5)
```



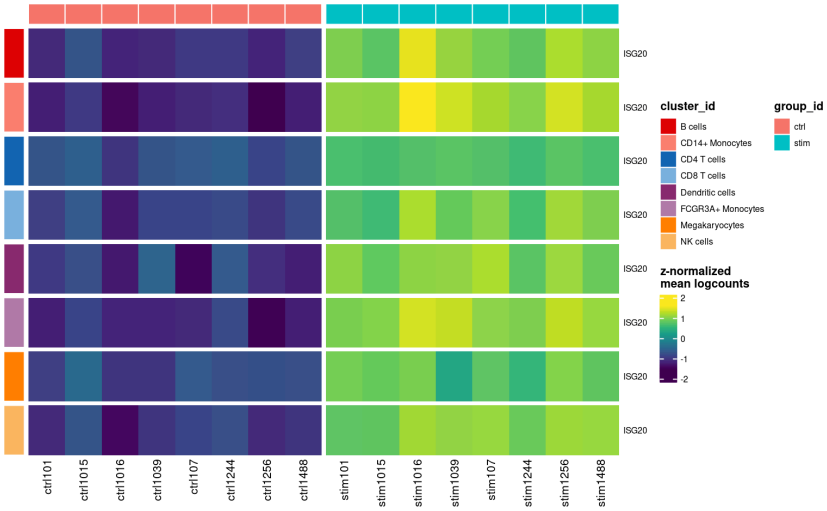
Alternatively, `pbHeatmap` provides a set of options regarding which cluster(s), gene(s), and comparison to include (arguments `k`, `g` and `c`, respectively). For example, the following options render a heatmap visualizing the top 20 DS genes for the *B cells* cluster:

```
# top-20 DS genes for single cluster
pbHeatmap(sce, res, k = "B cells")
```



Similarly, we can visualize the cluster-sample means of a single gene of interest across all clusters in order to identify cell-types that are affected similarly by different experimental conditions:

```
# single gene across all clusters
pbHeatmap(sce, res, g = "ISG20")
```



Session info

```
sessionInfo()
```

```
## R version 4.4.0 RC (2024-04-16 r86468)
## Platform: x86_64-pc-linux-gnu
## Running under: Ubuntu 22.04.4 LTS
##
## Matrix products: default
## BLAS:   /home/biocbuild/bbs-3.20-bioc/R/lib/libRblas.so
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.10.0
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_GB              LC_COLLATE=C
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## time zone: America/New_York
## tzcode source: system (glibc)
##
## attached base packages:
## [1] stats4    stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] UpSetR_1.4.0                scater_1.33.0
##  [3] scuttle_1.15.0              muscData_1.17.0
##  [5] SingleCellExperiment_1.27.0 SummarizedExperiment_1.35.0
##  [7] Biobase_2.65.0              GenomicRanges_1.57.0
##  [9] GenomeInfoDb_1.41.0         IRanges_2.39.0
## [11] S4Vectors_0.43.0            MatrixGenerics_1.17.0
## [13] matrixStats_1.3.0           ExperimentHub_2.13.0
## [15] AnnotationHub_3.13.0        BiocFileCache_2.13.0
## [17] dbplyr_2.5.0                BiocGenerics_0.51.0
## [19] purrr_1.0.2                 muscat_1.19.0
## [21] limma_3.61.0                ggplot2_3.5.1
## [23] dplyr_1.1.4                 cowplot_1.1.3
## [25] BiocStyle_2.33.0
##
## loaded via a namespace (and not attached):
##   [1] RcppAnnoy_0.0.22          splines_4.4.0
##   [3] filelock_1.0.3            bitops_1.0-7
##   [5] tibble_3.2.1              lifecycle_1.0.4
##   [7] Rdpack_2.6                edgeR_4.3.0
##   [9] doParallel_1.0.17         globals_0.16.3
##  [11] lattice_0.22-6            MASS_7.3-60.2
##  [13] backports_1.4.1           magrittr_2.0.3
##  [15] sass_0.4.9                rmarkdown_2.26
##  [17] jquerylib_0.1.4           yaml_2.3.8
##  [19] sctransform_0.4.1         DBI_1.2.2
##  [21] minqa_1.2.6               RColorBrewer_1.1-3
##  [23] multcomp_1.4-25           abind_1.4-5
##  [25] zlibbioc_1.51.0           EnvStats_2.8.1
##  [27] glmmTMB_1.1.9             TH.data_1.1-2
##  [29] rappdirs_0.3.3            sandwich_3.1-0
##  [31] circlize_0.4.16           GenomeInfoDbData_1.2.12
##  [33] ggrepel_0.9.5             pbkrtest_0.5.2
##  [35] irlba_2.3.5.1             listenv_0.9.1
##  [37] parallelly_1.37.1         DelayedMatrixStats_1.27.0
##  [39] codetools_0.2-20          DelayedArray_0.31.0
##  [41] tidyselect_1.2.1          shape_1.4.6.1
##  [43] farver_2.1.1              UCSC.utils_1.1.0
##  [45] lme4_1.1-35.3             ScaledMatrix_1.13.0
##  [47] viridis_0.6.5             jsonlite_1.8.8
##  [49] GetoptLong_1.0.5          BiocNeighbors_1.23.0
##  [51] survival_3.6-4            iterators_1.0.14
##  [53] emmeans_1.10.1            foreach_1.5.2
##  [55] tools_4.4.0               progress_1.2.3
##  [57] Rcpp_1.0.12               blme_1.0-5
##  [59] glue_1.7.0                gridExtra_2.3
##  [61] SparseArray_1.5.0         xfun_0.43
##  [63] mgcv_1.9-1                DESeq2_1.45.0
##  [65] withr_3.0.0               numDeriv_2016.8-1.1
##  [67] BiocManager_1.30.22       fastmap_1.1.1
##  [69] boot_1.3-30               fansi_1.0.6
##  [71] caTools_1.18.2            digest_0.6.35
##  [73] rsvd_1.0.5                mime_0.12
##  [75] R6_2.5.1                  estimability_1.5
##  [77] colorspace_2.1-0          Cairo_1.6-2
##  [79] gtools_3.9.5              RSQLite_2.3.6
##  [81] RhpcBLASctl_0.23-42       utf8_1.2.4
```

```
##  [83] tidyr_1.3.1               generics_0.1.3
##  [85] variancePartition_1.35.0  data.table_1.15.4
##  [87] corpcor_1.6.10            prettyunits_1.2.0
##  [89] httr_1.4.7                S4Arrays_1.5.0
##  [91] uwot_0.2.2                pkgconfig_2.0.3
##  [93] gtable_0.3.5              blob_1.2.4
##  [95] ComplexHeatmap_2.21.0     XVector_0.45.0
##  [97] remaCor_0.0.18            htmltools_0.5.8.1
##  [99] bookdown_0.39             TMB_1.9.11
## [101] clue_0.3-65               scales_1.3.0
## [103] png_0.1-8                 fANCOVA_0.6-1
## [105] knitr_1.46                reshape2_1.4.4
## [107] rjson_0.2.21              curl_5.2.1
## [109] coda_0.19-4.1             nlme_3.1-164
## [111] nloptr_2.0.3              cachem_1.0.8
## [113] zoo_1.8-12                GlobalOptions_0.1.2
## [115] stringr_1.5.1             BiocVersion_3.20.0
## [117] KernSmooth_2.23-22        parallel_4.4.0
## [119] vipor_0.4.7               AnnotationDbi_1.67.0
## [121] pillar_1.9.0              grid_4.4.0
## [123] vctrs_0.6.5               gplots_3.1.3.1
## [125] BiocSingular_1.21.0       beachmat_2.21.0
## [127] xtable_1.8-4              cluster_2.1.6
## [129] beeswarm_0.4.0            evaluate_0.23
## [131] magick_2.8.3              tinytex_0.50
## [133] mvtnorm_1.2-4             cli_3.6.2
## [135] locfit_1.5-9.9            compiler_4.4.0
## [137] rlang_1.1.3               crayon_1.5.2
## [139] future.apply_1.11.2       labeling_0.4.3
## [141] plyr_1.8.9                ggbeeswarm_0.7.2
## [143] stringi_1.8.3             viridisLite_0.4.2
## [145] BiocParallel_1.39.0       Biostrings_2.73.0
## [147] lmerTest_3.1-3            munsell_0.5.1
## [149] aod_1.3.3                 Matrix_1.7-0
## [151] hms_1.1.3                 bit64_4.0.5
## [153] sparseMatrixStats_1.17.0  future_1.33.2
## [155] KEGGREST_1.45.0           statmod_1.5.0
## [157] highr_0.10                rbibutils_2.2.16
## [159] memoise_2.0.1             broom_1.0.5
## [161] bslib_0.7.0               bit_4.0.5
```

References

Butler, Andrew, Paul Hoffman, Peter Smibert, Efthymia Papalexi, and Rahul Satija. 2018. “Integrating Single-Cell Transcriptomic Data Across Different Conditions, Technologies, and Species.” *Nature Biotechnology* 36 (5): 411–20.

Crowell, Helena L, Charlotte Soneson, Pierre-Luc Germain, Daniela Calini, Ludovic Collin, Catarina Raposo, Dheeraj Malhotra, and Mark D Robinson. 2020. “Muscat Detects Subpopulation-Specific State Transitions from Multi-Sample Multi-Condition Single-Cell Transcriptomics Data.” *Nature Communications* 11 (1): 6077.

Diaz-Mejia, J Javier, J Javier Diaz-Mejia, Elaine C Meng, Alexander R Pico, Sonya A MacParland, Troy Ketela, Trevor J Pugh, Gary D Bader, and John H Morris. 2019. “Evaluation of Methods to Assign Cell Type Labels to Cell Clusters from Single-Cell RNA-sequencing Data.” *F1000Research* 8: 296.

Duò, Angelo, Mark D Robinson, and Charlotte Soneson. 2018. “A Systematic Performance Evaluation of Clustering Methods for Single-Cell RNA-

seq Data.” *F1000Research* 7: 1141.

Freytag, Saskia, Luyi Tian, Ingrid Lönnstedt, Milica Ng, and Melanie Bahlo. 2018. “Comparison of Clustering Tools in R for Medium-Sized 10x Genomics Single-Cell RNA-sequencing Data.” *F1000Research* 7: 1297.

Hafemeister, Christoph, and Rahul Satija. 2019. “Normalization and Variance Stabilization of Single-Cell RNA-seq Data Using Regularized Negative Binomial Regression.” *bioRxiv* 576827.

Kang, Hyun Min, Meena Subramaniam, Sasha Targ, Michelle Nguyen, Lenka Maliskova, Elizabeth McCarthy, Eunice Wan, et al. 2018. “Multiplexed Droplet Single-Cell RNA-sequencing Using Natural Genetic Variation.” *Nature Biotechnology* 36 (1): 89–94.

Love, Michael I, Wolfgang Huber, and Simon Anders. 2014. “Moderated Estimation of Fold Change and Dispersion for RNA-seq Data with DESeq2.” *Genome Biology* 15 (12): 550.

McCarthy, Davis J, Kieran R Campbell, Aaron T L Lun, and Quin F Wills. 2017. “Scater: Pre-Processing, Quality Control, Normalization and Visualization of Single-Cell RNA-seq Data in R.” *Bioinformatics* 33 (8): 1179–86.

Ritchie, Matthew E, Belinda Phipson, Di Wu, Yifang Hu, Charity W Law, Wei Shi, and Gordon K Smyth. 2015. “Limma Powers Differential Expression Analyses for RNA-sequencing and Microarray Studies.” *Nucleic Acids Research* 43 (7): e47.

Robinson, Mark D, Davis J McCarthy, and Gordon K Smyth. 2010. “EdgeR: A Bioconductor Package for Differential Expression Analysis of Digital Gene Expression Data.” *Bioinformatics* 26 (1): 139–40.

Soneson, Charlotte, and Mark D Robinson. 2018. “Bias, Robustness and Scalability in Single-Cell Differential Expression Analysis.” *Nature Methods* 15 (4): 255–61.

Stegle, Oliver, Sarah A Teichmann, and John C Marioni. 2015. “Computational and Analytical Challenges in Single-Cell Transcriptomics.” *Nature Reviews Genetics* 16 (3): 133–45.

Stuart, Tim, Andrew Butler, Paul Hoffman, Christoph Hafemeister, Efthymia Papalexi, William M Mauck 3rd, Yuhan Hao, Marlon Stoeckius, Peter Smibert, and Rahul Satija. 2019. “Comprehensive Integration of Single-Cell Data.” *Cell* 177 (7): 1888–1902.e21.

Trapnell, Cole. 2015. “Defining Cell Types and States with Single-Cell Genomics.” *Genome Research* 25 (10): 1491–8.

Wagner, Allon, Aviv Regev, and Nir Yosef. 2016. “Revealing the Vectors of Cellular Identity with Single-Cell Genomics.” *Nature Biotechnology* 34 (11): 1145–60.

Zhang, Allen W, Ciara O, Elizabeth Chavez, Jamie L P Lim, Andrew McPherson, Matt Wiens, Pascale Walters, et al. 2019. “Probabilistic Cell Type Assignment of Single-Cell Transcriptomic Data Reveals Spatiotemporal Microenvironment Dynamics in Human Cancers.” *bioRxiv* 521914.