BASiCS workflow: a step-by-step analysis of expression variability using single cell RNA sequencing data

Alan O'Callaghan*[1], Nils Eling[2], John C. Marioni[3,4], and Catalina A. Vallejos†[1,5]

[1]MRC Human Genetics Unit, Institute of Genetics & Molecular Medicine, University of Edinburgh, Western General Hospital, Crewe Road, Edinburgh, EH4 2XU, UK
[2]Department of Quantitative Biomedicine, University of Zurich, Winterthurerstrasse 190, CH-8057, Zurich, Switzerland
[3]European Molecular Biology Laboratory, European Bioinformatics Institute, Wellcome Trust Genome Campus, Hinxton, Cambridge CB10 1SD, UK
[4]Cancer Research UK Cambridge Institute, University of Cambridge, Li Ka Shing Centre, Cambridge, CB2 0RE, UK
[5]The Alan Turing Institute, British Library, 96 Euston Road, London, NW1 2DB, UK

Abstract Cell-to-cell gene expression variability is an inherent feature of complex biological systems, such as immunity and development. Single-cell RNA sequencing is a powerful tool to quantify this heterogeneity, but it is prone to strong technical noise. In this article, we describe a step-by-step computational workflow that uses the BASiCS Bioconductor package to robustly quantify expression variability within and between known groups of cells (such as experimental conditions or cell types). BASiCS uses an integrated framework for data normalisation, technical noise quantification and downstream analyses, whilst propagating statistical uncertainty across these steps. Within a single seemingly homogeneous cell population, BASiCS can identify highly variable genes that exhibit strong heterogeneity as well as lowly variable genes with stable expression. BASiCS also uses a probabilistic decision rule to identify changes in expression variability between cell populations, whilst avoiding confounding effects related to differences in technical noise or in overall abundance. Using a publicly available dataset, we guide users through a complete pipeline that includes preliminary steps for quality control, as well as data exploration using the scater and scran Bioconductor packages. The workflow is accompanied by a Docker image that ensures the reproducibility of our results.

Keywords

Single-cell RNA sequencing, expression variability, transcriptional noise, differential expression testing

*a.b.o'callaghan@sms.ed.ac.uk
†catalina.vallejos@igmm.ed.ac.uk

Introduction

Single-cell RNA-sequencing (scRNA-seq) enables the study of genome-wide transcriptional heterogeneity in cell populations that is not captured by bulk experiments [1, 2, 3]. On the broadest level, this heterogeneity can reflect the presence of distinct cell subtypes or states. Alternatively, it can be due to gradual changes along biological processes, such as development and differentiation. Several clustering and pseudotime inference methods have been developed to characterise these types of heterogeneity [4, 5]. However, there is a limited availability of computational tools tailored to study more subtle variability within seemingly homogeneous cell populations. This variability can reflect deterministic or stochastic events that regulate gene expression and, among others, has been reported to increase prior to cell fate decisions [6] as well as during ageing [7].

Stochastic variability within a seemingly homogeneous cell population — often referred to as transcriptional *noise* — can arise from intrinsic and extrinsic sources [8, 9]. Extrinsic noise refers to stochastic fluctuations induced by different dynamic cellular states (e.g. cell cycle, metabolism, intra/inter-cellular signalling) [10, 11, 12]. In contrast, intrinsic noise arises from stochastic effects on biochemical processes such as transcription and translation [8]. Intrinsic noise can be modulated by genetic and epigenetic modifications (such as mutations, histone modifications, CpG island length and nucleosome positioning) [13, 14, 15] and usually occurs at the gene level [8]. Cell-to-cell gene expression variability estimates derived from scRNA-seq data capture a combination of these effects, as well as deterministic regulatory mechanisms [9]. Moreover, these variability estimates can also be inflated by the technical noise that is typically observed in scRNA-seq data [16].

Different strategies have been incorporated into scRNA-seq protocols to control or attenuate technical noise. For example, external RNA spike-in molecules (such as the set introduced by the External RNA Controls Consortium, ERCC [17]) can be added to each cell's lysate in a (theoretically) known fixed quantity. Spike-ins can assist quality control steps [18], data normalisation [19] and can be used to infer technical noise [16]. Another strategy is to tag individual cDNA molecules using unique molecular identifiers (UMIs) before PCR amplification [20]. Reads that contain the same UMI can be collapsed into a single molecule count, attenuating technical variability associated to cell-to-cell differences in amplification and sequencing depth (these technical biases are not fully removed unless sequencing to saturation [19]). However, despite the benefits associated to the use of spike-ins and UMIs, these are not available for all scRNA-seq protocols [21].

The Bioconductor package *BASiCS* implements a Bayesian hierarchical framework that accounts for both technical and biological sources of noise in scRNA-seq datasets [22, 23, 24]. *BASiCS* jointly performs data normalisation, technical noise quantification and downstream analyses, whilst propagating statistical uncertainty across these steps. These features are implemented within a probabilistic model that builds upon a negative binomial framework, a widely used distribution in the context of bulk and scRNA-seq experiments [25, 26, 27]. Critically, *BASiCS* enables the quantification of transcriptional variability within a population of cells, while accounting for the overall mean-variance relationship that typically arises in scRNA-seq data [28]. Furthermore, when available, *BASiCS* can also leverage extrinsic spike-in molecules to aid data normalisation.

This article complements existing scRNA-seq workflows based on the Bioconductor ecosystem (e.g. [29, 30]), providing a detailed framework for transcriptional variability analyses using *BASiCS*. We describe a step-by-step workflow that uses *scater* [18] and *scran* [29] to perform quality control (QC) as well as initial exploratory analyses. Our analysis pipeline includes practical guidance to assess the convergence of the Markov Chain Monte Carlo (MCMC) algorithm that is used to infer model parameters in *BASiCS*, as well as recommendations to interpret and post-process the model outputs. Finally, through a case study in the context of immune cells, we illustrate how *BASiCS* can be used to identify highly and lowly variable genes within a cell population, as well as to compare expression profiles between experimental conditions or cell types.

All source code used to generate the results presented in this article is available on Github. To ensure the reproducibility of this workflow, the analysis environment and all software dependencies are provided as a Docker image [31]. The image can be obtained from Docker Hub.

Methods

This step-by-step scRNA-seq workflow is primarily based on the Bioconductor package ecosystem [32]. A graphical overview is provided in Figure 1 and its main components are described below.

Input data

```
library("SingleCellExperiment")
```

We use *SingleCellExperiment* to convert an input matrix of raw read-counts (molecule counts for UMI-based protocols) into a `SingleCellExperiment` object that can also store its associated metadata, such as gene- and cell-specific information. Moreover, when available, the same object can also store read-counts for spike-in molecules (see `?altExp`). A major advantage of using a `SingleCellExperiment` object as the input for scRNA-seq analyses is the interoperability across a large number of Bioconductor packages [32].

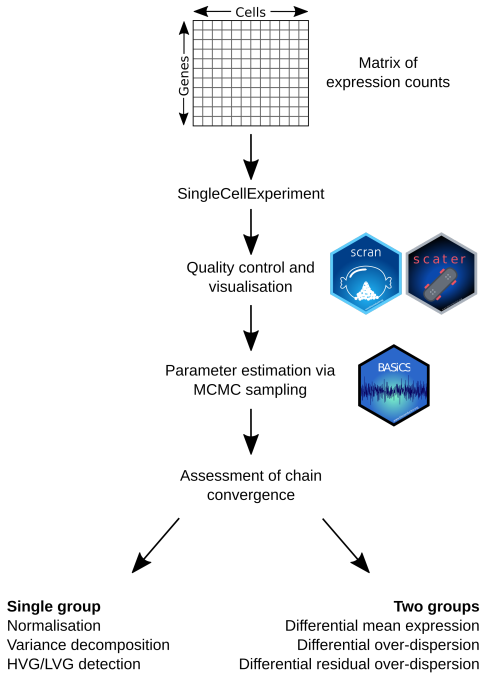

Figure 1. Graphical overview for the scRNA-seq analysis workflow described in this manuscript. Starting from a matrix of expression counts, we use the scater and scran Bioconductor packages to perform QC and initial exploratory analyses. To robustly quantify transcriptional heterogeneity within seemingly homogeneous cell populations, we apply the BASiCS Bioconductor package and illustrate how BASiCS can be used to analyse a single or multiple pre-specified groups of cells.

QC and exploratory data analysis

```
library("scater")
library("scran")
```

An critical step in scRNA-seq analyses is QC, removing low quality samples that may distort downstream analyses. In this step, we use QC diagnostics to identify and remove samples that correspond to broken cells, that are empty, or that contain multiple cells [33]. We also typically remove lowly expressed genes that represent less reliable information. The *OSCA* online book provides an extensive overview on important aspects of how to perform QC of scRNA-seq data, including exploratory analyses [32].

Here, we use the *scater* package [18] to calculate QC metrics for each cell (e.g. total read-count) and gene (e.g. percentage of zeroes across all cells), respectively. Moreover, we use the visualisation tools implemented in *scater* to explore the input dataset and its associated QC diagnostic metrics. For further exploratory data analysis we use the *scran* package [29]. The latter can perform *global scaling* normalisation, calculating cell-specific scaling factors that capture global differences in read-counts across cells (e.g. due to sequencing depth and PCR amplification) [34]. To quantify transcriptional variability, *scran* can be used to infer an overall trend between mean expression and the squared coefficent of variation (CV^2) for each gene. To derive variability estimates that are not confounded by this overall trend, *scran* also defines gene-specific DM (distance to the mean) estimates as the distance between CV^2 and a rolling median along the range of mean expression values [35]. DM estimates enable exploratory analyses of cell-to-cell gene expression variability, but a measure of statistical uncertainty is not readily available for these estimates. As such, gene-specific downstream inference (such as differential variability testing) is precluded.

BASiCS - Bayesian Analysis of Single Cell Sequencing data

```
library("BASiCS")
```

The *BASiCS* package uses a Bayesian hierarchical framework that borrows information across all genes and cells to robustly quantify transcriptional variability [36]. Similar to the approach adopted in *scran*, *BASiCS* infers cell-specific global scaling normalisation parameters. However, instead of inferring these as a pre-processing step, *BASiCS* uses an integrated approach wherein data normalisation and downstream analyses are performed simultaneously, thereby propagating statistical uncertainty. To quantify technical noise, the original implementation of *BASiCS* uses information from extrinsic spike-in molecules as control features, but the model has been extended to address situations wherein spike-ins are not available [28].

BASiCS summarises the expression pattern for each gene through gene-specific *mean* and *over-dispersion* parameters. Mean parameters μ_i quantify the overall expression for each gene i across the cell population under study. In contrast, δ_i captures the excess of variability that is observed with respect to what would be expected in a homogeneous cell population, beyond technical noise. *BASiCS* uses δ_i as a proxy to quantify transcriptional variability. To account for the strong relationship that is typically observed between gene-specific mean expression and over-dispersion estimates, Eling *et al.* [28] introduced a joint prior specification for these parameters. This joint prior formulation has been observed to improve posterior inference when the data is less informative (e.g. small sample size, lowly expressed genes), borroring information across all genes to infer an overall trend that captures the relationship between mean and over-dispersion. The trend is then used to derive gene-specific *residual over-dispersion* parameters ϵ_i that are not confounded by mean expression. Similar to DM values implemented in *scran*, these are defined as deviations with respect to the overall trend.

Within a population of cells, *BASiCS* decomposes the total observed variability in expression measurements into technical and biological components [22]. This enables the identification of *highly variable genes* (HVGs) that capture the major sources of heterogeneity within the analysed cells [16]. HVG detection is often used as feature selection, to identify the input set of genes for subsequent analyses. *BASiCS* can also highlight *lowly variable genes* (LVGs) that exhibit stable expression across the population of cells. These may relate to essential cellular functions and can assist the development of new data normalisation or integration strategies [37].

BASiCS also provides a probabilistic decision rule to perform differential expression analyses between two (or more) pre-specified groups of cells [23, 28]. While several differential expression tools have been proposed for scRNA-seq data (e.g. [38, 39]), some evidence suggests that these do not generally outperform popular bulk RNA-seq tools [40]. Moreover, most of these methods are only designed to uncover changes in overall expression, ignoring the more complex patterns that can arise at the single cell level [41]. Instead, *BASiCS* embraces the high granularity of scRNA-seq data, uncovering changes in cell-to-cell expression variability that are not confounded by differences in technical noise or in overall expression.

Case study: analysis of naive CD4$^+$ T cells

As a case study, we use scRNA-seq data generated for CD4$^+$ T cells using the C1 Single-Cell Auto Prep System (Fluidigm®). Martinez-Jimenez *et al.* profiled naive (hereafter also referred to as unstimulated) and activated (3 hours using *in vitro* antibody stimulation) CD4$^+$ T cells from young and old animals across two mouse strains to study changes in expression variability during ageing and upon immune activation [7]. They extracted naive or effector memory CD4$^+$ T cells from spleens of young or old animals, obtaining purified populations using either magnetic-activated cell sorting (MACS) or fluorescence activated cell sorting (FACS). External ERCC spike-in RNA [17] was added to aid the quantification of technical variability across all cells and all experiments were performed in replicates (hereafter also referred to as batches).

Downloading the data

The matrix with raw read counts can be obtained from ArrayExpress under the accession number E-MTAB-4888. In the matrix, column names contain library identifiers and row names display gene Ensembl identifiers.

```
if (!file.exists("downloads/"))
  dir.create("downloads", showWarnings = FALSE)
if (!file.exists("downloads/raw_data.txt")) {
  website <- "https://www.ebi.ac.uk/arrayexpress/files/E-MTAB-4888/"
  file <- "E-MTAB-4888.processed.1.zip"
  download.file(
    paste0(website, file),
    destfile = "downloads/raw_data.txt.zip"
  )
  unzip("downloads/raw_data.txt.zip", exdir = "downloads")
  file.remove("downloads/raw_data.txt.zip")
}

CD4_raw <- read.table("downloads/raw_data.txt", header = TRUE, sep = "\t")
CD4_raw <- as.matrix(CD4_raw)
```

The input matrix contains data for 1,513 cells and 31,181 genes (including 92 ERCC spike-ins). Information about experimental conditions and other metadata is available under the same accession number.

```
if (!file.exists("downloads/metadata_file.txt")) {
  website <- "https://www.ebi.ac.uk/arrayexpress/files/E-MTAB-4888"
  file <- "E-MTAB-4888.additional.1.zip"
  download.file(
    paste0(website, file),
    destfile = "downloads/metadata.txt.zip"
  )
  unzip("downloads/metadata.txt.zip", exdir = "downloads")
  file.remove("downloads/metadata.txt.zip")
}

CD4_metadata <- read.table(
  "downloads/metadata_file.txt",
  header = TRUE,
  sep = "\t"
)

# Save sample identifiers as rownames
rownames(CD4_metadata) <- CD4_metadata$X
```

The columns in the metadata file contain library identifiers (`X`), strain information (`Strain`; *Mus musculus castaneus* or *Mus musculus domesticus*), the age of the animals (`Age`; young or old), stimulation state of the cells (`Stimulus`; naive or activated), batch information (`Individuals`; associated to different mice), and cell type information (`Celltype`; via FACS or MACS purification).

Here, we convert the data and metadata described above into a `SingleCellExperiment` object. For this purpose, we first separate the input matrix of expression counts into two matrices associated to intrinsic genes and external spike-ins, respectively. Within the `SingleCellExperiment` object, the latter is stored separately as an *alternative experiment* (see `?altExp`).

```
# Separate intrinsic from ERCC counts
bio_counts <- CD4_raw[!grepl("ERCC", rownames(CD4_raw)), ]
spike_counts <- CD4_raw[grepl("ERCC", rownames(CD4_raw)), ]
# Generate the SingleCellExperiment object
sce_CD4_all <- SingleCellExperiment(
  assays = list(counts = as.matrix(bio_counts)),
  colData = CD4_metadata[colnames(CD4_raw), ]
)
# Add read-counts for spike-ins as an alternative experiment
altExp(sce_CD4_all, "spike-ins") <- SummarizedExperiment(
  assays = list(counts = spike_counts)
)
```

Hereafter, our analysis focuses on naive and activated CD4$^+$ T cells obtained from young *Mus musculus domesticus* animals, purified using MACS-based cell sorting. Here, we extract these 146 samples.

```
ind_select <- sce_CD4_all$Strain == "Mus musculus domesticus" &
  sce_CD4_all$Age == "Young" &
  sce_CD4_all$Celltype == "MACS-purified Naive"
sce_naive_active <- sce_CD4_all[, ind_select]
sce_naive_active
```

```
## class: SingleCellExperiment
## dim: 31089 146
## metadata(0):
## assays(1): counts
## rownames(31089): ENSMUSG00000000001 ENSMUSG00000000003 ...
##   ENSMUSG00000106668 ENSMUSG00000106670
## rowData names(0):
## colnames(146): do6113 do6118 ... do6493 do6495
## colData names(6): X Strain ... Individuals Celltype
## reducedDimNames(0):
## altExpNames(1): spike-ins
```

Annotation

Input read counts were annotated using Ensembl gene identifiers. In order to facilitate the visualisation and interpretation of results, it is often useful to generate a mapping from Ensembl gene IDs to gene symbols using the BioMart software suite (http://www.biomart.org) via the Bioconductor package, *biomaRt* [42]. These packages can also be used to obtain gene-pathways mappings and other information such as gene length, useful for performing functional analysis of the gene sets identified in downstream analyses.

```
if(!file.exists("rds/"))
  dir.create("rds", showWarnings = FALSE)

library(biomaRt)

if (!file.exists("rds/genenames.rds")) {
  # Initialize mart and dataset
  ensembl <- useMart(
    biomart = "ensembl",
    dataset = "mmusculus_gene_ensembl"
  )

  # Select gene ID and gene name
  genenames <- getBM(
    attributes = c("ensembl_gene_id", "external_gene_name"),
    mart = ensembl
  )

  rownames(genenames) <- genenames$ensembl_gene_id
  saveRDS(genenames, "rds/genenames.rds")
} else {
  genenames <- readRDS("rds/genenames.rds")
}
```

We add this information as `rowData` within the `SingleCellExperiment` object created above.

```
# Merge biomaRt annotation
my.rowdata <- data.frame(ensembl_gene_id = rownames(sce_naive_active))
my.rowdata <- merge(my.rowdata, genenames, by = "ensembl_gene_id", all.x = TRUE)
rownames(my.rowdata) <- rownames(sce_naive_active)
# Check to see that the order is correct after merge
# Sum must be equal to zero
sum(my.rowdata$ensembl_gene_id != rownames(my.rowdata))
```

```
## [1] 0
```

```
# Add to the SingleCellExperiment object
rowData(sce_naive_active) <- my.rowdata
```

QC and exploratory data analysis

The data available at E-MTAB-4888 have been already filtered to remove poor quality samples. The QC applied in [7] removed cells with: (i) fewer than 1,000,000 total reads, (ii) less than 20% of reads mapped to endogenous genes, (iii) less than 1,250 or more than 3,000 detected genes and (iv) more than 10% or fewer than 0.5% of reads mapped to mitochondrial genes. As an illustration, we visualise some of these metrics. We also include another widely used QC diagnostic plot that compares the total number (or fraction) of spike-in counts versus the total number (or fraction) of endogeneous counts. In such a plot, low quality samples are characterised by a high fraction of spike-in counts and a low fraction of endogeneous counts (see Figure 2).

```
# Calculate and plot per cell QC metrics
sce_naive_active <- addPerCellQC(sce_naive_active, use_altexps = TRUE)
p_cellQC1 <- plotColData(
  sce_naive_active,
  x = "sum",
  y = "detected") +
  xlab("Total engogenous reads per cell") +
```

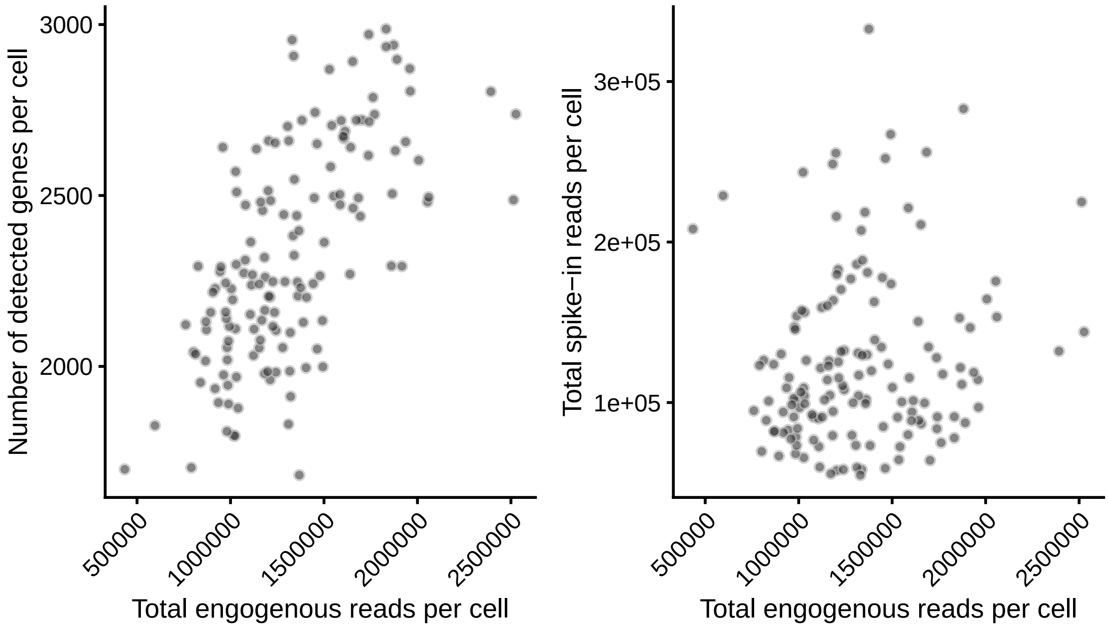

Figure 2. Cell-level QC metrics. The total number of endogenous read-counts (excludes non-mapped and intronic reads) is plotted against the total number of detected genes (left) and the total number of spike-in read-counts (right).

```
  ylab("Number of detected genes per cell") +
  theme(axis.text.x = element_text(hjust = 1, angle = 45))
p_cellQC2 <- plotColData(
  sce_naive_active,
  x = "sum",
  y = "altexps_spike-ins_sum") +
  xlab("Total engogenous reads per cell") +
  ylab("Total spike-in reads per cell") +
  theme(axis.text.x = element_text(hjust = 1, angle = 45))

library(patchwork)
p_cellQC1 + p_cellQC2
```

We can also visualise these metrics with respect to cell-level metadata, such as the experimental conditions (active vs unstimulated) and the different mice that cells were collected from (see Figure 3).

```
p_stimulus <- plotColData(
    sce_naive_active,
    x = "sum",
    y = "detected",
    colour_by = "Stimulus"
  ) +
  xlab("Total engogenous reads per cell") +
  ylab("Number of detected genes per cell") +
  theme(
    legend.position = "bottom",
    axis.text.x = element_text(angle = 45, hjust = 1)
  )
p_batch <- plotColData(
    sce_naive_active,
    x = "sum",
    y = "detected",
    colour_by = "Individuals"
  ) +
  xlab("Total engogenous reads per cell") +
  ylab("Number of detected genes per cell") +
  theme(
```

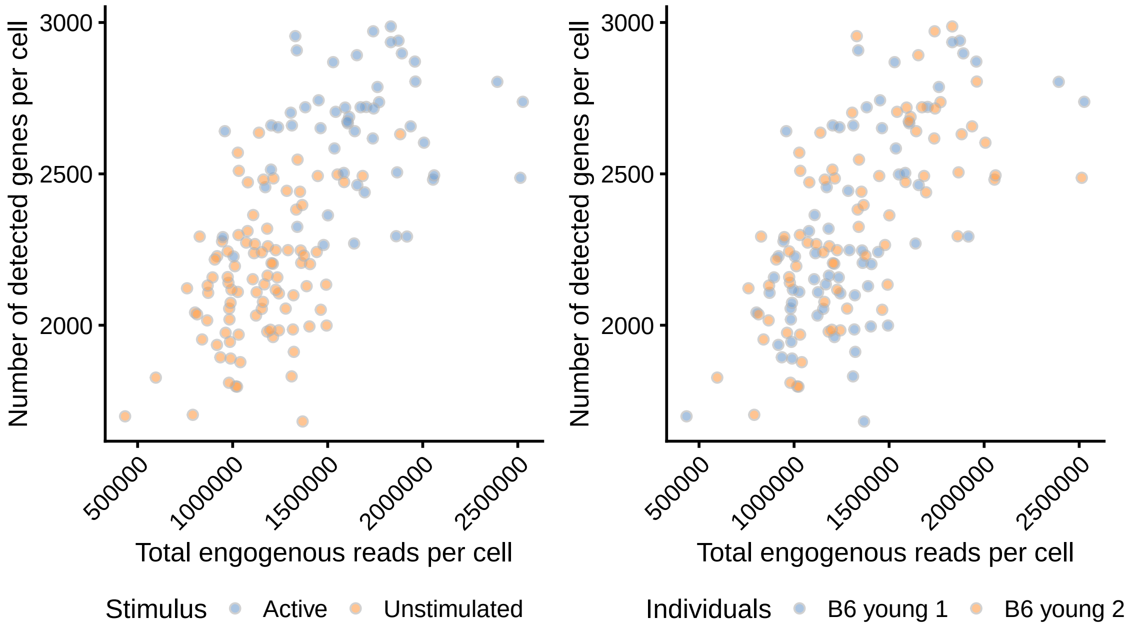

Figure 3. Cell-level QC metrics according to cell-level metadata. The total number of endogenous reads (excludes non-mapped and intronic reads) is plotted against the total number of detected genes. Colour indicates the experimental condition (left) and animal of origin (right) for each cell.

```
    legend.position = "bottom",
    axis.text.x = element_text(angle = 45, hjust = 1)
  )
p_stimulus + p_batch
```

To further explore the underlying structure of the data, we compute global scaling normalisation factors using *scran* and perform a principal component analysis (PCA) of log-transformed normalised expression counts using *scater*. As seen in Figure 4, this analysis suggests the absence of strong batch effects. It should be noted that *scran* normalisation is not strictly necessary in the *BASiCS* workflow and we only use it here as part of the exploratory data analysis. Moreover, count-based models for dimensionality reduction (e.g. [27, 43]) could be used as an alternative to PCA, removing the need for log normalisation .

```
# Global scaling normalisation + log tranformation + PCA
sce_naive_active <- computeSumFactors(sce_naive_active)
sce_naive_active <- logNormCounts(sce_naive_active)
sce_naive_active <- runPCA(sce_naive_active)
p_stimulus <- plotPCA(sce_naive_active, colour_by = "Stimulus") +
  theme(legend.position = "bottom")
p_batch <- plotPCA(sce_naive_active, colour_by = "Individuals") +
  theme(legend.position = "bottom")
p_stimulus + p_batch
```

In addition to cell-specific QC, we also recommend the use of a gene filtering step prior to the use of *BASiCS*. The purpose of this filter is to remove lowly expressed genes that were largely undetected through sequencing, making reliable variability estimates difficult to obtain. Here, we remove all genes that are not detected in at least 5 cells across both experimental conditions or that have an average read count below 1. These thresholds can vary across datasets and should be informed by gene-specific QC metrics such as those shown in Figure 5.

```
# Calculate per gene QC metrics
sce_naive_active <- addPerFeatureQC(sce_naive_active, exprs_values = "counts")
# Remove genes with zero total counts across all cells
sce_naive_active <- sce_naive_active[rowData(sce_naive_active)$detected != 0, ]
# Transform 'detected' metadata into number of cells
rowData(sce_naive_active)$detected_cells <-
  rowData(sce_naive_active)$detected * ncol(sce_naive_active) / 100
# Define inclusion criteria for genes
rowData(sce_naive_active)$include_gene <- rowData(sce_naive_active)$mean >= 1 &
```

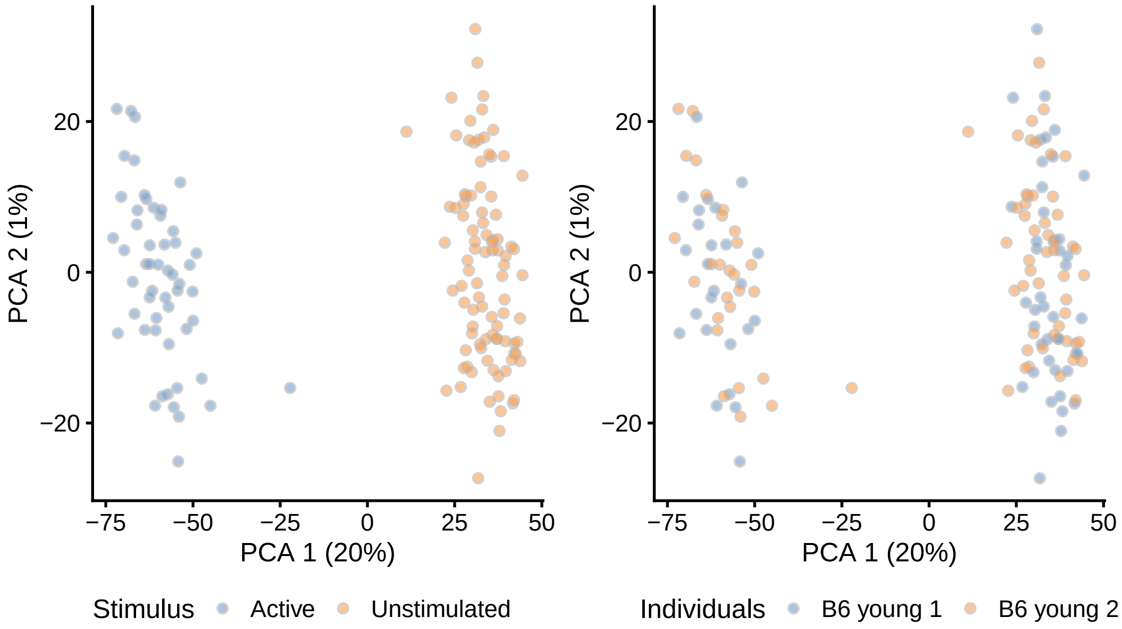

Figure 4. First two principal components of log-transformed expression counts after scran normalisation. Colour indicates the experimental condition (left) and animal of origin (right) for each cell.

```
  rowData(sce_naive_active)$detected_cells >= 5
plotRowData(
    sce_naive_active,
    x = "detected_cells",
    y = "mean",
    colour_by = "include_gene") +
  xlab("Total engogenous reads per cell") +
  ylab("Number of detected genes per cell") +
  scale_x_log10() +
  scale_y_log10() +
  theme(
    legend.position = "bottom",
    axis.text.x = element_text(angle = 45, hjust = 1)
  ) +
geom_vline(xintercept = 5, linetype = "dashed", col = "grey60") +
geom_hline(yintercept = 1, linetype = "dashed", col = "grey60")
```

```
# Apply gene filter
sce_naive_active <- sce_naive_active[rowData(sce_naive_active)$include_gene, ]
```

Subsequently, we also require users to remove spike-in molecules that were not captured through sequencing. We will do this separately for naive and active cells separately.

```
ind_active <- sce_naive_active$Stimulus == "Active"
ind_naive <- sce_naive_active$Stimulus == "Unstimulated"
spikes <- assay(altExp(sce_naive_active))
detected_spikes_active <- rowSums(spikes[, ind_active] > 0) > 0
detected_spikes_naive <- rowSums(spikes[, ind_naive] > 0) > 0
detected_spikes <- detected_spikes_naive & detected_spikes_active
altExp(sce_naive_active) <- altExp(sce_naive_active)[detected_spikes, ]
```

The final dataset used in subsequent analyses contains 146 cells, 8953 genes and 49 spike-ins.

Input data for BASiCS

Here, we apply the *BASiCS* model separately to cells from each experimental condition (93 naive and 53 activated cells). We create separate `SingleCellExperiment` objects for each group of cells.

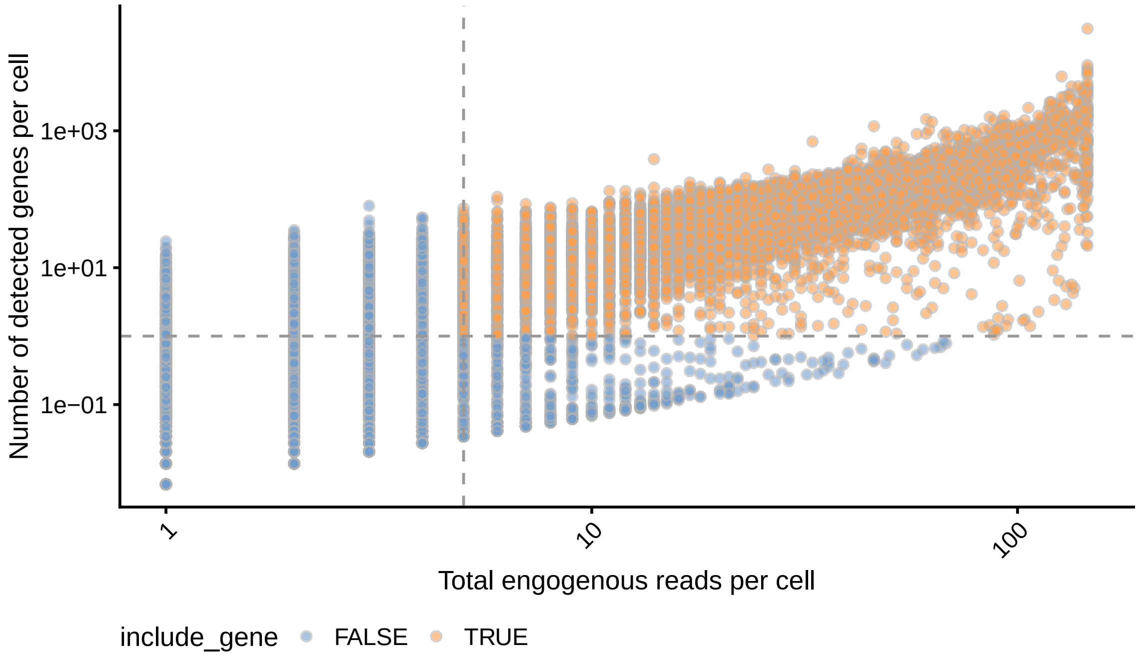

Figure 5. Average read-count for each gene is plotted against the number of cells in which that gene was detected. Dashed grey lines are shown at the thresholds below which genes are removed.

```
sce_naive <- sce_naive_active[, ind_naive]
sce_active <- sce_naive_active[, ind_active]
```

BASiCS requires the user to update these objects with additional information, using a specific format. Firstly, if multiple batches of sequenced cells are available (e.g. multiple donors that cells were extracted from, or multiple sequencing batches from the same experimental condition), this information must be included under the `BatchInfo` label as part of the cell-level metadata.

```
colData(sce_naive)$BatchInfo <- colData(sce_naive)$Individuals
colData(sce_active)$BatchInfo <- colData(sce_active)$Individuals
```

If spike-ins will be used to aid data normalisation and technical noise quantification, *BASiCS* also requires the number of spike-in molecules that were added to each well. For each spike-in gene i, this corresponds to:

$$\mu_i = C_i \times 10^{-18} \times (6.022 \times 10^{23}) \times V \times D \qquad \text{where,}$$

- C_i is the concentration for the spike-in i (measured in $aM\mu l^{-1}$),
- V is the volume added into each well (measure in nl) and
- D is a dilution factor.

The remaining factors in the equation above are unit conversion constants (e.g. from moles to molecules). For the CD4^{+} T cell data, the authors added a 1:50,000 dilution of the ERCC spike-in mix 1 and a volume of 9nl was added into each well (see `https://www.fluidigm.com/faq/ifc-9`). Finally, input concentrations C_i can be downloaded from https://assets.thermofisher.com/TFS-Assets/LSG/manuals.

```
if (!file.exists("downloads/spike_info.txt")) {
  website <- "https://assets.thermofisher.com/TFS-Assets/LSG/manuals"
  file <- "cms_095046.txt"
  download.file(
    paste0(website, file),
    destfile = "downloads/spike_info.txt"
  )
}

ERCC_conc <- read.table("downloads/spike_info.txt", sep = "\t", header = TRUE)
```

Based on this information, the calculation above proceeds as follows

```
# Moles per micro litre
ERCC_mmul <- ERCC_conc$concentration.in.Mix.1..attomoles.ul. * (10^(-18))
# Molecule count per micro litre (1 mole comprises 6.02214076 x 10^{23} molecules)
ERCC_countmul <- ERCC_mmul * (6.02214076 * (10^23))
# Application of the dilution factor (1:50,000)
ERCC_count <- ERCC_countmul / 50000
# Multiplying by the volume added into each well
ERCC_count_final <- ERCC_count * 0.009
```

To update the `sce_naive` and `sce_active` objects, the user must create a `data.frame` whose first column contains the labels associated to the spike-in molecule (e.g. ERCC-00130) and whose second column contains the input number of molecules calculated above. We add this information as metadata for `altExp(sce_naive)` and `altExp(sce_active)`, respectively.

```
SpikeInput <- data.frame(
  Names = ERCC_conc$ERCC.ID,
  count = ERCC_count_final
)
# Exclude spike-ins not included in the input SingleCellExperiment objects
# and ensure the order of the rows is identical
SpikeInput <- SpikeInput[match(rownames(altExp(sce_naive)), SpikeInput$Names), ]

metadata(sce_naive)$SpikeInput <- SpikeInput
metadata(sce_active)$SpikeInput <- SpikeInput
```

Parameter estimation using BASiCS

Parameter estimation is implemented in the `BASiCS_MCMC` function using an adaptive Metropolis within Gibbs algorithm [44]. The primary inputs for `BASiCS_MCMC` correspond to:

- `Data`: a `SingleCellExperiment` object created as described in the previous sections.
- `N`: the total number of MCMC iterations.
- `Thin`: thining period for output storage (only the `Thin`-th MCMC draw is stored).
- `Burn`: the initial number of MCMC iterations to be discarded.
- `Regression`: if TRUE a join prior is assigned to μ_i and δ_i [28], and residual over-dispersion values ϵ_i are inferred. Alternatively, independent log-normal priors are assigned to μ_i and δ_i [23].
- `PriorMu`: If `"EmpiricalBayes"`, μ_i's are assigned a log-normal prior with gene-specific location hyper-parameters defined via an empirical Bayes framework. Alternatively, if `PriorMu = "default"`, location hyper-parameter are set to be equal 0 for all genes.
- `WithSpikes`: if `TRUE` information from spike-in molecules is used to aid data normalisation and to quantify technical noise.

As a default, we recommend to use `Regression = TRUE` as we have observed that the joint prior introduced by Eling *et al.* leads to more stable, particularly for small sample sizes and lowly expressed genes. Moreover, the joint prior formulation enables users to obtain a measure of transcriptional variability that is not confounded by mean expression. We also recommend to use `PriorMu = "EmpiricalBayes"` as we have observed that the empirical Bayes framework [REF] improves estimation performance for sparser datasets. Additional optional parameters can be used to store the generated output (`StoreChains`, `StoreDir`, `RunName`) and to monitor the progress of the algorithm (`PrintProgress`).

Here, we run the MCMC sampler separately for naive and activated cells. We use 40,000 iterations, discarding the initial 20,000 iterations. We recommend this setting as a default choice, as we have observed it to ensure good convergence for the algorithm across multiple datasets. However, for large datasets and less sparse datasets, a lower number of iterations may be sufficient. Practical guidance about the diagnostics criteria required to assess the performance of the MCMC algorithn is provided in the next section.

```r
chain_naive <- BASiCS_MCMC(
  Data = sce_naive,
  N = 40000,
  Thin = 20,
  Burn = 20000,
  Regression = TRUE,
  PriorMu = "EmpiricalBayes",
  WithSpikes = TRUE,
  StoreChains = TRUE,
  StoreDir = "rds/",
  RunName = "naive"
)

chain_active <- BASiCS_MCMC(
  Data = sce_active,
  N = 40000,
  Thin = 20,
  Burn = 20000,
  Regression = TRUE,
  PriorMu = "EmpiricalBayes",
  WithSpikes = TRUE,
  StoreChains = TRUE,
  StoreDir = "rds/",
  RunName = "active"
)
```

This first of these samplers takes 127 minutes to complete on a 3.4 GHz Intel Core i7 4770k procesor with 16GB RAM, while the second takes 75 minutes. For convenience, these can be obtained online at https://git.ecdf.ed.ac.uk/vallejosgroup/basicsworkflow2020.

```r
if (!file.exists("rds/chain_naive.Rds")) {
  website <- "https://git.ecdf.ed.ac.uk/vallejosgroup/basicsworkflow2020/raw/master/"
  file <- "chain_naive.Rds"
  download.file(
    paste0(website, file),
    destfile = "rds/chain_naive.Rds"
  )
}
if (!file.exists("rds/chain_active.Rds")) {
  website <- "https://git.ecdf.ed.ac.uk/vallejosgroup/basicsworkflow2020/raw/master/"
  file <- "chain_active.Rds"
  download.file(
    paste0(website, file),
    destfile = "rds/chain_active.Rds"
  )
}
chain_naive <- readRDS("rds/chain_naive.Rds")
chain_active <- readRDS("rds/chain_active.Rds")
```

The output from `BASiCS_MCMC` is a `BASiCS_Chain` object that contains the draws associated to all model parameters (as `N` = 40,000, `Thin` = 20 and `Burn` = 20,000, the object contains 1,000 draws for each parameter). These can be accessed using the `displayChainBASiCS` function. For example, the following code displays the first 2 draws for mean expression parameters μ_i associated to the first 3 genes.

```r
displayChainBASiCS(chain_naive, Param = "mu")[1:2, 1:3]
```

```
##      ENSMUSG00000000001 ENSMUSG00000000056 ENSMUSG00000000085
## [1,]           97.48793           38.17646           3.484543
## [2,]          162.83523           18.13590           2.503296
```

MCMC diagnostics

Before interpreting the parameter estimates obtained by *BASiCS*, it is critical to assess the convergence of the MCMC algorithm, i.e. whether the MCMC reached its stationary distribution. If convergence has been

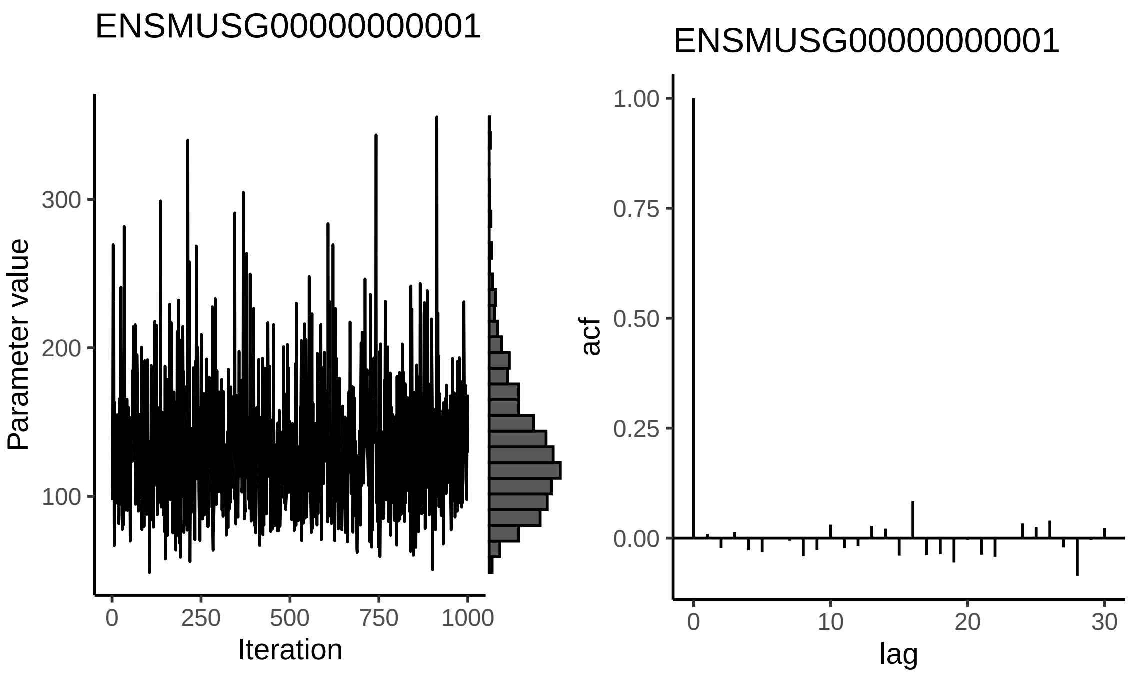

Figure 6. Trace plot, marginal histogram, and autocorrelation function for a gene in naive cells following MCMC sampling. Trace plots should explore the posterior well, without getting stuck in one location or drifting over time towards a region of higher density. High autocorrelation indicates that the number of effective independent samples is low. It is good practice to perform these visualisation for many different parameters; here we only show one.

achieved, each parameter should not (on average) evolve significantly across iterations, and MCMC draws are expected to be stochastic fluctuations around a horizontal trend. Generally, it is not possible to prove convergence but multiple graphical and quantitative convergence diagnostics have been proposed to assess the lack of convergence (e.g. [45, 46]). Some advocate the use of multiple MCMC chains using different starting values in order to ensure that the algorithm consistently converges to the same distribution. For *BASiCS*, we have observed that using informed starting values (e.g. based on *scran* normalisation) and a sufficiently large value for `N` and `Burn` generally leads to consistent across multiple MCMC runs. Hence, the focus of this section is to evaluate quantitative measures of convergence (e.g. [47]) based on a single MCMC chain.

Traceplots can be used to visually assess the history of MCMC iterations for a specific parameter (e.g. Figure 6). As mentioned above, significan departures from a horizontal trend suggest a lack of convergence. As illustrated in Figure 6, histograms can also be used to display the marginal distribution for each parameter. Users should expect these to follow a unimodal distribution. Failure to satisfy these graphical diagnostics suggest that `N` and `Burn` must be increased. Alternatively, more stringent quality control could be applied to the input data, as genes with very low counts often suffer from slow convergence and poor sampling efficiency.

```
plot(chain_naive, Param = "mu", Gene = 1)
```

As *BASiCS* infers thousands of parameters, it is impractical to assess these diagnostics for each parameter. Thus, it is helpful to use numerical diagnostics, applied to multiple parameters simultaneously. Here, we focus on the diagnostic criterion proposed by Geweke [47] which compares the average draws obtained during the initial (10% after burn in, by default) and the final part of the chain (50% by default). Large absolute Z-scores suggest that the algorithm has not converged. For the naive and activatived CD4$^+$ T datasets most Z-scores associated to mean expression parameters μ_i were small in aboslute value (see Figure **??**).

```
library("coda")
library("ggplot2")
library("viridis")

# Calculate and plot Geweke Z scores
gew_mu_naive <- geweke.diag(mcmc(displayChainBASiCS(chain_naive, Param = "mu")))$z
gew_mu_active <- geweke.diag(mcmc(displayChainBASiCS(chain_active, Param = "mu")))$z

myplot_params <- list(
  geom_hex(),
```

```
  geom_hline(yintercept = c(-3, 3), col = "firebrick", linetype = "dashed"),
  scale_fill_viridis(),
  labs(x = "log(mu)", y = "Geweke Z-score"))

p_geweke_naive <- ggplot() +
  aes(
    log10(colMedians(displayChainBASiCS(chain_naive, Param = "mu"))),
    gew_mu_naive
  ) +
  myplot_params +
  ggtitle("Naive cells")
p_geweke_active <- ggplot() +
  aes(
    log10(colMedians(displayChainBASiCS(chain_active, Param = "mu"))),
    gew_mu_active
  ) +
  myplot_params+
  ggtitle("Activated cells")

p_geweke_naive + p_geweke_active
```

As well as assessing MCMC convergence, it is important to ensure that the MCMC algorithm has efficiently explored the parameter space. For example, the autocorrelation function (e.g. Figure 6, right panel) can be used to quantify the correlation between the chain and its lagged versions. Strong autocorrelation indicates that neighbouring MCMC samples are highly dependent and suggest poor sampling efficiency. The latter may indicate that the MCMC draws do not contain sufficient information to produce accurate posterior estimates. In other words, highly correlated MCMC samplers require more samples to produce the same level of Monte Carlo error for an estimate (defined as the variance of a Monte Carlo estimate across repetitions [48]).

The effective sample size (ESS) is a related measure which represents a proxy for the number of independent draws generated by the MCMC sampler [49]. The latter is defined as:

$$\text{ESS} = \frac{N_{tot}}{1 + 2\sum_{k=1}^{\infty} \rho(k)},$$

where N_{tot} represents the total number of MCMC draws (after burn-in and thining) and $\rho(k)$ $\rho(k)$ is the auto-correlation at lag k. ESS estimates associated to mean expression parameters for the naive and activated CD4$^+$ T cells are displayed in Figure **??**. Whilst ESS is around 1,000 (N_{tot} in this case) for most genes, we observe low ESS values for a small proportion of genes (primarily lowly expressed genes whose expression was only captured in a small number of cells). As described later in this manuscript, `BASiCS_TestDE` automatically excludes genes with low ESS when performing differential expression testing.

```
ess_mu_naive <- BASiCS_DiagPlot(chain_naive, Param = "mu") +
  theme(legend.position = "bottom")
ess_mu_active <- BASiCS_DiagPlot(chain_active, Param = "mu") +
  theme(legend.position = "bottom")
ess_mu_naive + ess_mu_active
```

Quantifying transcriptional variability using BASiCS

Studying gene-level transcriptional variability can provide insights about the regulation of gene expression, and how it relates to the properties of genomic features (e.g. CpG island composition [15]), transcriptional dynamics [50] and aging [7], among others. The squared coefficient of variation (CV^2) is widely used as a proxy for transcriptional variability. For example, here we obtain CV^2 estimates for each gene using *scran* normalised expression counts as input. Instead, *BASiCS* infers transcriptional variability using gene-specific over-dispersion parameters δ_i (see *Methods*). Here, we compare these approaches focusing on naive CD4$^+$ T cells (the analysis for active cells shows similar results). As seen in Figure 7A, CV^2 and posterior estimates for δ_i are highly correlated. Moreover, both variability metrics are confounded by differences in mean expression, i.e. highly expressed genes tend to exhibit lower variability (Figures 7B-C). To remove this confounding, *scran* and *BASiCS* derive *residual variability* estimates as deviations with respect to an global mean-variability trend (see *Methods*). These are derived using the DM approach [35] and the residual over-dispersion parameters ϵ_i defined by [28], respectively. For the naive CD4$^+$ T data, both approaches led to strongly correlated estimates (Figure 7D) and, as expected, neither DM values or posterior estimates for ϵ_i are seen to be associated with mean expression (Figure 7E-F). However, unlike the DM method, the integrated approach implemented in *BASiCS* provides a direct measure of statistical uncertainty for these estimates. As illustrated in the following sections, this enables gene-specific downstream inference such as differential variability analyses.

```r
library("ggpointdensity")
library("viridis")

# Get BASiCS posterior estimates for mean and variability - naive cells
summary_naive <- Summary(chain_naive)
parameter_df <- data.frame(
  mu = displaySummaryBASiCS(summary_naive, Param = "mu")[, 1],
  delta = displaySummaryBASiCS(summary_naive, Param = "delta")[, 1],
  epsilon = displaySummaryBASiCS(summary_naive, Param = "epsilon")[, 1]
)

# Get scran estimates for mean and variability - naive cells
sce_naive <- logNormCounts(sce_naive, log = FALSE)
parameter_df$mean_scran <- rowMeans(assay(sce_naive, "normcounts"))
parameter_df$cv2_scran <- rowVars(assay(sce_naive, "normcounts")) /
  parameter_df$mean_scran^2
parameter_df$DM <- DM(
  mean = parameter_df$mean_scran,
  cv2 = parameter_df$cv2_scran
)

# Remove genes without counts in > 2 cells - BASiCS estimates not provided
ind_not_na <- !(is.na(parameter_df$epsilon))

myplot_params <- list(geom_pointdensity(),
                      scale_colour_viridis(),
                      theme(
                        text = element_text(size=rel(3)),
                        legend.position = "bottom",
                        legend.text = element_text(angle = 45, size = 8),
                        legend.key.size =  unit(0.8, "cm")))

g1 <- ggplot(parameter_df[ind_not_na, ], aes(log10(cv2_scran), log10(delta))) +
  myplot_params +
  xlab("scran CV^2 (log10)") +
  ylab("BASiCS\nover-dispersion (log10)") +
  geom_abline(slope = 1, intercept = 0, col = "red")

g2 <- ggplot(parameter_df[ind_not_na, ], aes(log10(mean_scran), log10(cv2_scran))) +
  myplot_params +
  xlab("scran means (log10)") +
  ylab("scran CV^2 (log10)")

g3 <- ggplot(parameter_df[ind_not_na, ], aes(log10(mu), log10(delta))) +
  myplot_params +
  xlab("BASiCS means (log10)") +
  ylab("BASiCS\nover-dispersion (log10)")

g4 <- ggplot(parameter_df[ind_not_na, ], aes(DM, epsilon)) +
  myplot_params +
  xlab("scran DM") +
  ylab("BASiCS residual\nover-dispersion") +
  geom_abline(slope = 1, intercept = 0, col = "red")

g5 <- ggplot(parameter_df[ind_not_na, ], aes(log10(mean_scran), DM)) +
  myplot_params +
  xlab("scran means (log10)") +
  ylab("scran DM") +
  geom_hline(yintercept = 0, col = "red")

g6 <- ggplot(parameter_df[ind_not_na, ], aes(log10(mu), epsilon)) +
  myplot_params +
  xlab("BASiCS means (log10)") +
  ylab("BASiCS residual\nover-dispersion") +
  geom_hline(yintercept = 0, col = "red")
```

```
(g1 + g2 + g3) / (g4 + g5 + g6) +
  plot_annotation(tag_levels = "A") & theme(plot.tag = element_text(size = 15))
```

HVG/LVG detection using BASiCS

In *BASiCS*, the functions `BASiCS_DetectHVG` and `BASiCS_DetectLVG` can be used to identify genes with substantially high (HVG) or low (LVG) transcriptional variability within a population of cells. If the input `BASiCS_Chain` object was generated by `BASiCS_MCMC` with `Regression = TRUE` (recommended setting), this analysis is based on the posterior distribution obtained for gene-specific residual over-dispersion parameters ϵ_i (alternatively, the approach introduced by [22] can be used). HVGs are marked as those for which ϵ_i exceeds a pre-defined threshold with high probability, where the probability cut-off is chosen to match a given expected false discovery rate (EFDR; default EFDR = 10%) [51]. A similar approach is implemented for LVG detection, but based on whether ϵ_i is below a pre-specified threshold. In principle, a threshold for ϵ_i could be directly specified by the user. For example, if the threshold for ϵ_i is equal to log(2), HVGs would be those genes for which the over-dispersion is estimated to be at least 2 times higher than would be expected given the inferred mean expression level. In practice, however, it may be of interest to rank genes and to select the those with the highest or the lowest residual over-dispersion. For this purpose, the `PercentileThreshold` parameter can be used to define the threshold based on the empirical distribution of posterior estimates for residual over-dispersion parameters ϵ_i. For example, if `PercentileThreshold = 0.9`, the `BASiCS_DetectHVG` function defines the threshold as the 90% percentile of the above distribution.

```
# Highly variable genes
HVG <- BASiCS_DetectHVG(chain_naive, PercentileThreshold = 0.9)
# Lowly variable genes
LVG <- BASiCS_DetectLVG(chain_naive, PercentileThreshold = 0.1)

ggplot(HVG@Table) +
  geom_point(aes(log10(Mu), Epsilon), na.rm = TRUE, colour = "gray") +
  xlab("BASiCS means (log10)") +
  ylab("BASiCS residual\nover-dispersion") +
  geom_hline(yintercept = 0, lty = 2) +
  geom_point(data = HVG@Table[HVG@Table$HVG == TRUE, ],
             aes(log10(Mu), Epsilon), na.rm = TRUE, colour = "red") +
  geom_point(data = LVG@Table[LVG@Table$LVG == TRUE, ],
             aes(log10(Mu), Epsilon), na.rm = TRUE, colour = "blue")
```

For the naive CD4$^+$ T data, we obtained *343* HVG and *716* LVG. As shown in Figure 8, these genes are distributed across a wide range of mean expression values. As an illustration, Figure 9 shows the distribution of normalised expression values for selected HVG and LVG, focusing on examples with similar mean expression levels (Figure 9A). As expected HVG tend to exhibit a wider distribution and potentially bimodal distribution (Figure 9B). Instead, LVG tend to have more narrow and unimodal distributions (Figure 9C).

```
library(reshape2)

# Obtain normalised expression values
DC_naive <- BASiCS_DenoisedCounts(sce_naive, chain_naive)

HVG_Table <- merge(HVG@Table, genenames,
                   by.x = "GeneName", by.y = "ensembl_gene_id",
                   all.x = TRUE, sort = FALSE)
LVG_Table <- merge(LVG@Table, genenames,
                   by.x = "GeneName", by.y = "ensembl_gene_id",
                   all.x = TRUE, sort = FALSE)

# Select HVG/LVG genes with similar mean expression values
low.exp <- 2.8
up.exp <- 3
HVG1 <- HVG_Table[HVG_Table$HVG == TRUE &
                    log10(HVG_Table$Mu) > low.exp &
                    log10(HVG_Table$Mu) < up.exp,]
LVG1 <- LVG_Table[LVG_Table$LVG == TRUE &
                    log10(LVG_Table$Mu) > low.exp &
```

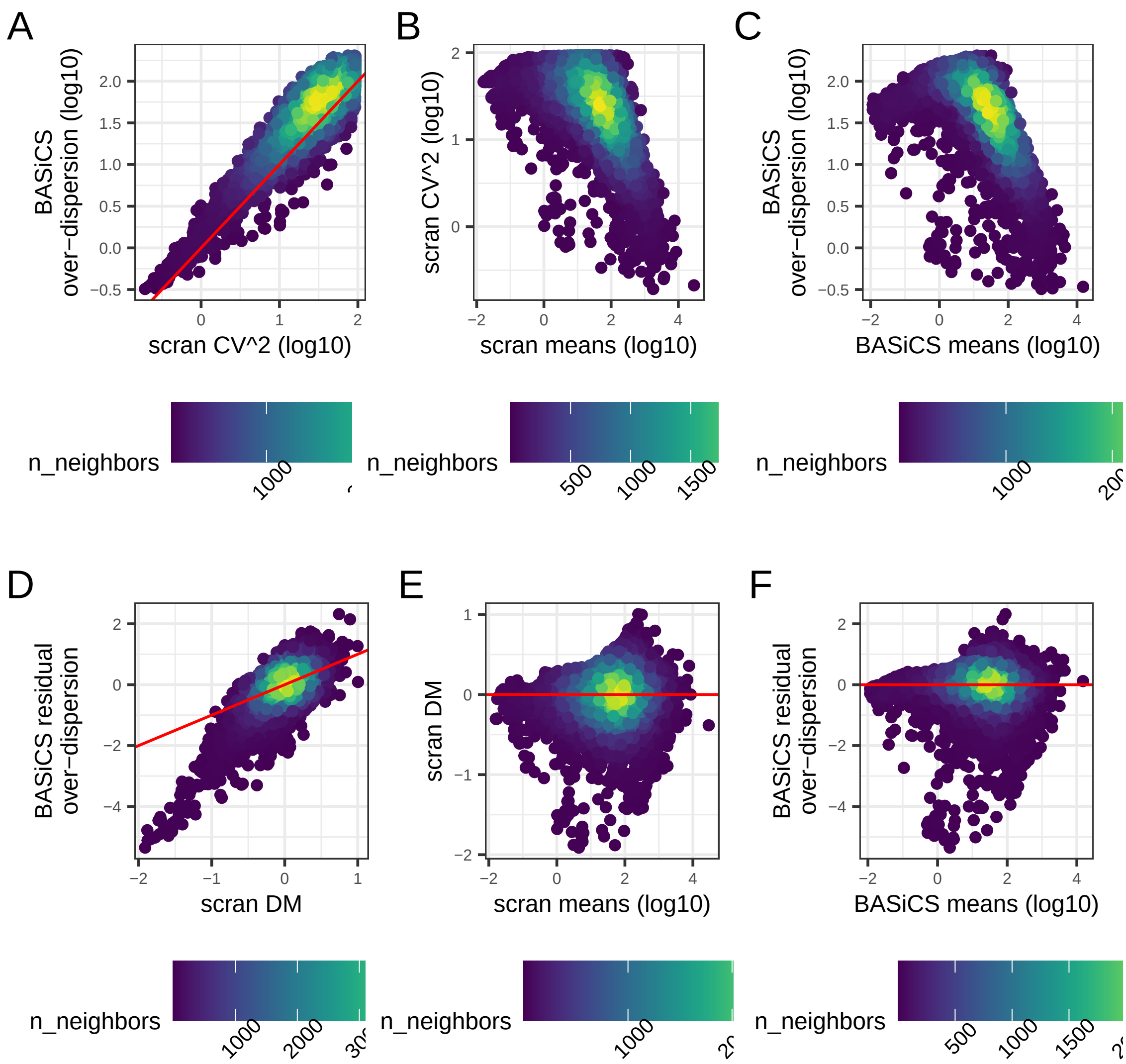

Figure 7. Comparison of gene-specific transcriptional variability estimates and mean expression estimates obtained for each gene using BASiCS and scran. For this analysis, we exclude genes that are not expressed in at least 2 cells. BASiCS estimates for each gene are defined by the posterior median of the associated parameter. scran estimates for each gene are derived after applying the pooling normalisation strategy proposed by Lun et al. Points are coloured according to the local density of genes along the x- and y- axis. A: scran squared CV estimates versus BASiCS estimates for over-dispersion parameters. B: scran estimates for mean expression and the squared CV. C: BASiCS estimates for mean expression and over-dispersion parameters. D: BASiCS estimates for residual over-dispersion parameters versus DM values estimated by scran. E: scran estimates for mean expression and DM values. F: BASiCS estimates for mean expression and residual over-dispersion parameters.

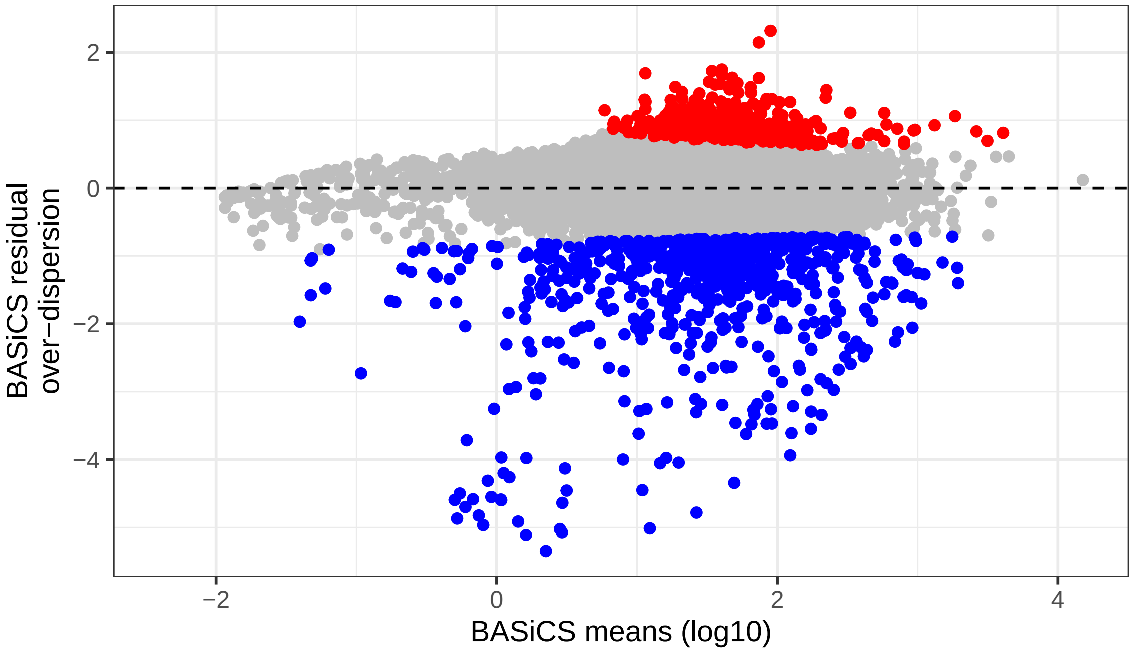

Figure 8. HVG and LVG detection using BASiCS. For each gene, BASiCS posterior estimates (posterior medians) associated to mean expression and residual over-dispersion parameters are plotted. Genes are coloured according to HVG/LVG status. Genes that are not expressed in at least 2 cells are excluded.

```
                    log10(LVG_Table$Mu) < up.exp,]

# Order by epsilon and select top 3 HVG and LVG within the genes selected above
HVG1 <- HVG1[order(HVG1$Epsilon, decreasing = TRUE),]
LVG1 <- LVG1[order(LVG1$Epsilon, decreasing = FALSE),]
topHVG <- log10(t(DC_naive[HVG1$GeneName[1:3], ]) + 1)
topLVG <- log10(t(DC_naive[LVG1$GeneName[1:3], ]) + 1)
# Add genenames
colnames(topHVG) <- HVG1$external_gene_name[1:3]
colnames(topLVG) <- LVG1$external_gene_name[1:3]

myplot_params <- list(geom_violin(na.rm = TRUE),
                      coord_flip(),
                      ylim(-0.05, max(log10(DC_naive + 1))),
                      geom_jitter(position=position_jitter(0.3)),
                      ylab("log10(norm count + 1)"),
                      xlab("Gene"))
plot_hvg <- ggplot(melt(topHVG), aes(x = Var2, y = value)) +
  myplot_params + ggtitle("Example HVG")

plot_lvg <- ggplot(melt(topLVG), aes(x = Var2, y = value)) +
  myplot_params + ggtitle("Example LVG")

plot_hvg + plot_lvg + plot_annotation(tag_levels = "A")
```

Differential mean and variability testing using BASiCS

This section highlights the use of *BASiCS* to perform differential expression tests for mean and variability between different pre-specified populations of cells and experimental conditions. Here, we compare the naive CD4$^+$ T cells, analysed in the previous section, to activated CD4$^+$ T cells obtained by the same authors [7]. Naive CD4$^+$ T cells were activated for 3 hours using plate-bound CD3e and CD28 antibodies. T cell activation is linked to strong transcriptional shifts and the up-regulation of lineage specific marker genes, such as Tbx21 and Gata1 [52, 53]. To generate this data, the authors did not add cytokines, which are needed for T cell differentiation [54]. Therefore, any heterogeneity in the activated cell population does not arise from cells residing in different lineage-specific differentiation states.

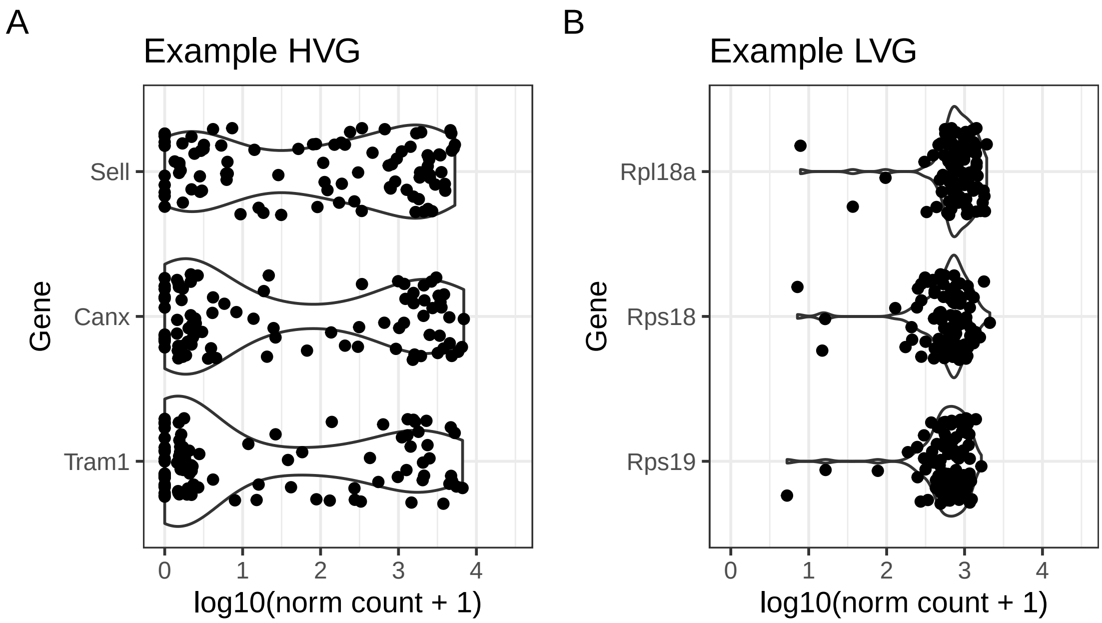

Figure 9. BASiCS denoised counts for example HVG and LVG with similar overall levels of expression.

BASiCS implements differential expression testing via the `BASiCS_TestDE` function. The main input parameters for `BASiCS_TestDE` are:

- `Chain1` and `Chain2`: two `BASiCS_Chain` objects created via the `BASiCS_MCMC` function. Each object corresponds to a different pre-specified group of cells
- `EpsilonM` and `EpsilonR`: introduce a minimum effect size (in a $\log_2$ fold change scale) for the detection of changes in mean or residual over-dispersion, respectively. This enables us to discard small expression changes that are less biologically meaningful. By default, we set these thresholds to be equivalent to a 50% change between the groups. However, different thresholds may be required depending on the context. For example, if most genes show strong differences in mean expression, it can be beneficial to increase the value of `EpsilonM` to focus on strong changes in mean expression.
- `EFDR_M` and `EFDR_R`: define the target EFDR to calibrate the decision rule associated to changes in mean or residual over-dispersion, respectively. Default is 10%.

- `MinESS`: Minimum ESS threshold beyond which genes will be excluded from the differential expression tests. This is used to increase the robustness of the results, excludes genes for which the sampler explored the parameter space less efficiently (see *MCMC diagnostics* Section). Default: `MinESS` = 100.

```
# Perform differential testing
Test_DE <- BASiCS_TestDE(
  Chain1 = chain_naive,
  Chain2 = chain_active,
  GroupLabel1 = "Naive",
  GroupLabel2 = "Active",
  Plot = FALSE,
  PlotOffset = FALSE
)
TableMean <- format(Test_DE, Which = "Mean", Filter = FALSE)
TableResDisp <- format(Test_DE, Which = "ResDisp", Filter = FALSE)
```

After running the test, we can now visualise the results in form of MA-plots (log ratio M versus mean average A) and volcano plots (posterior probability versus log ratio). For our analysis, these are displayed in Figure @ref(visualise_DE_plot).

```
(BASiCS_PlotDE(Test_DE, Parameters = "Mean", Plots = "MA") +
    BASiCS_PlotDE(Test_DE, Parameters = "Mean", Plots = "Volcano")) /
  (BASiCS_PlotDE(Test_DE, Parameters = "ResDisp", Plots = "MA") +
     BASiCS_PlotDE(Test_DE, Parameters = "ResDisp", Plots = "Volcano"))
```

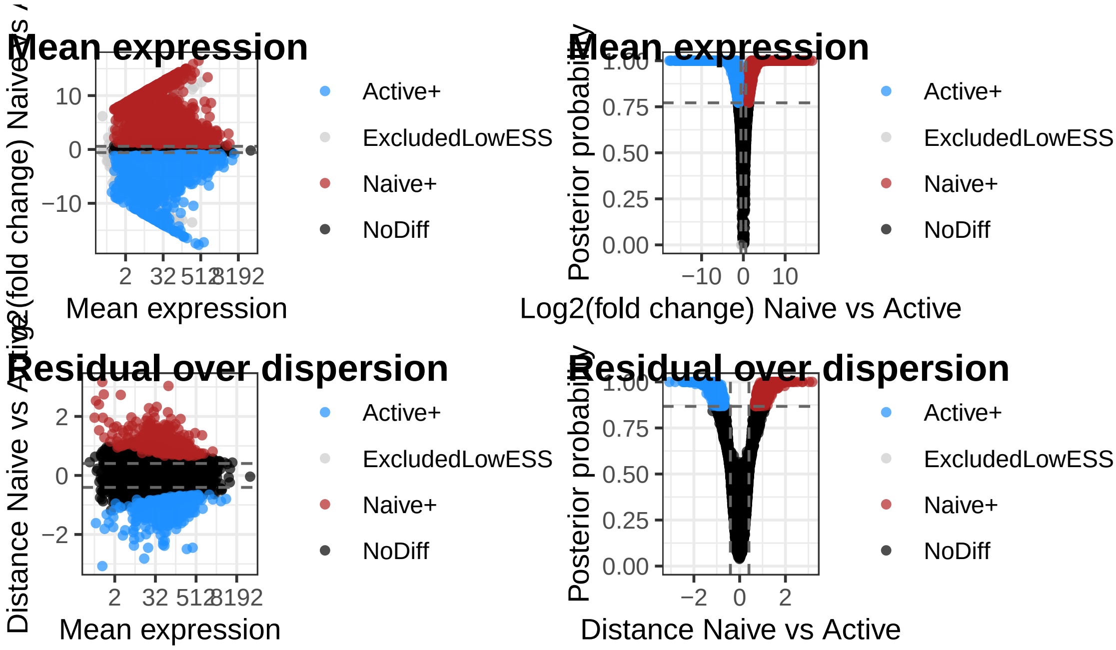

Figure 10. Fold changes of average expression in naive cells relative to active cells are plotted again mean expression. Colour indicates genes that were excluded from differential expression test, and those with significantly higher mean expression in either group.

It may also be useful to perform functional enrichment analysis to identify biologically meaningful patterns amongst differentially expressed genes. For example, this could be done using *goseq* [55]. We do not perform this here, as it is outside of the scope of *BASiCS*. However, a workflow for this process is detailed in Supplementary section TODO.

When interpreting the results of differential expression tests, it is useful to visualise expression patterns for differentially expressed genes in order to appraise the significance of the results and guide interpretation. First, here we focus on genes with changes in mean expression. For example, our test identifies a significant different in mean expression for Cd69 (ENSMUSG00000030156), a known marker of T cell activation [56] (Figures @ref(violin_plot_cd69)).

```
# Expression of Cd69 in both conditions
ind_cd <- which(rowData(sce_naive_active)$external_gene_name == "Cd69")
TableMean[TableMean$GeneName == rowData(sce_naive_active)$ensembl_gene_id[ind_cd],]
plotExpression(sce_naive_active,
               features = genenames[ind_cd, 1],
               x = "Stimulus"
)
```

It may also useful to visualise multiple genes at once. This is useful in quality checking the results of a differential expression analysis, and may also aid in identifying systematic patterns among differentially expressed genes and guiding downstream analysis. Here we use *ComplexHeatmap* to visualise genes that are up-regulated in each condition [57], the output of which is shown in Figure **??**.

```
library("ComplexHeatmap")
library("circlize")
library("RColorBrewer")

# Up-regulated in naive
ind_n <- TableMean$ResultDiffMean == "Naive+"
naive_mean <- TableMean[ind_n, ]
naive_mean <- naive_mean[order(naive_mean$MeanLog2FC, decreasing = FALSE), ]
naive_mean$Symbol <- genenames[naive_mean$GeneName, 2]

# Up-regulated in active
ind_a <- TableMean$ResultDiffMean == "Active+"
active_mean <- TableMean[ind_a, ]
active_mean <- active_mean[order(active_mean$MeanLog2FC, decreasing = TRUE), ]
```

```r
active_mean$Symbol <- genenames[active_mean$GeneName, 2]

heatmap_ngenes <- 15
heatmap_seq <- seq_len(heatmap_ngenes)

## Select genes with largest probability of differential expression
active_ind <- order(active_mean$ProbDiffMean, decreasing = TRUE)[heatmap_seq]
active_mean <- active_mean[active_ind, ]
## subset counts from each cell type to these genes
act_counts_act <- counts(sce_active)[active_mean$GeneName, ]
nai_counts_act <- counts(sce_naive)[active_mean$GeneName, ]

## Select genes with largest probability of differential expression
s_ind <- order(naive_mean$ProbDiffMean, decreasing = TRUE)[heatmap_seq]
naive_mean <- naive_mean[s_ind, ]
## subset counts from each cell type to these genes
act_counts_nai <- counts(sce_active)[naive_mean$GeneName, ]
nai_counts_nai <- counts(sce_naive)[naive_mean$GeneName, ]

## Calculate max absolute value for min/max of colour scale
all_mean <- c(
  active_mean$Mean1, active_mean$Mean2,
  naive_mean$Mean1, naive_mean$Mean2
)
max_range <- log(max(abs(all_mean)))
## colour scale symmetric around zero
colour <- colorRamp2(seq(-max_range, max_range, length.out = 9),
                     rev(brewer.pal(9, "RdBu")))

## Combine count matrices by cell type
counts_active <- rbind(act_counts_act, act_counts_nai)
counts_naive <- rbind(nai_counts_act, nai_counts_nai)

## split heatmaps by gene category
split <- data.frame(
  Upregulated = c(
    rep("Up-regulated \nin active", nrow(act_counts_act)),
    rep("Up-regulated \nin naive", nrow(act_counts_nai))
  )
)
syms <- genenames[rownames(counts_active), 2]
fontsize <- 7

Heatmap(
  log10(counts_naive + 1),
  row_labels = syms,
  row_names_gp = gpar(fontsize = fontsize),
  name = "log10(count + 1)",
  column_dend_height = unit(0.2, "npc"),
  column_title_side = "bottom",
  column_title = "Naive cells",
  show_column_names = FALSE,
  cluster_rows = FALSE,
  split = split,
  right_annotation = rowAnnotation(
    log_mu = log(c(active_mean$Mean1, naive_mean$Mean1)),
    col = list(log_mu = colour)
  ),
  col = viridis(100)) +
  Heatmap(
    log10(counts_active + 1),
    row_labels = syms,
    column_dend_height = unit(0.2, "npc"),
    row_names_gp = gpar(fontsize = fontsize),
    column_title = "Active cells",
```

```
    column_title_side = "bottom",
    show_column_names = FALSE,
    name = "log10(count + 1)",
    split = split,
    right_annotation = rowAnnotation(
      log_mu = log(c(active_mean$Mean2, naive_mean$Mean2)),
      col = list(log_mu = colour)
    ),
    cluster_rows = FALSE,
    col = viridis(100))
```

While other computational tools exist to perform differential mean expression analysis, we next want to highlight the use of *BASiCS* for differential variability testing. To avoid the mean versus over-dispersion confounding, we recommend this analysis to use residual over-dispersion parameters ϵ_i as input.

While one could focus on the gene sets that show significant changes in residual over-dispersion, here, we want to highlight how to analyse changes in mean expression in parallel to changes in variability. For this, we will first combine the results of the differential mean expression and the differential residual over-dispersion test.

```
TableCombined <- merge(TableMean, TableResDisp, by = "GeneName")
```

While the analysis in the previous section is well suited to detect global changes in variability (e.g. detecting if one cell population overall displays higher expression variabilty), it does not allow the testing of changes in mean expression and expression variability in parallel. For this, *BASiCS* compares the residual over-dispersion parameters, which do not scale with mean expression, between the two conditions. Here, we remove genes that are lowly expressed in both conditions.

```
# exclude lowly expressed genes
high_expr <- TableMean$Mean1 > 1 & TableMean$Mean2 > 1
```

We can now visualise the changes in residual over-dispersion between naive and activated $CD4^+$ T cells in the form of a MA-plot (Figure 11). In this visualisation, the difference between the posterior medians of the residual over-dispersion parameters ϵ are shown on the y-axis. Epsilon values for genes that are not expressed in at least 2 cells per conditions are marked as `NA` and are therefore not being displayed.

```
TableResDisp <- format(Test_DE, Which = "ResDisp", Filter = FALSE)
BASiCS_PlotDE(Test_DE, Parameters = "ResDisp", Plot = "MA")
```

References

[1] Oliver Stegle, Sarah a. Teichmann, and John C. Marioni. Computational and analytical challenges in single-cell transcriptomics. *Nature Reviews Genetics*, 16(3):133–145, jan 2015. ISSN 1471-0056. doi: 10.1038/nrg3833. URL http://www.nature.com/doifinder/10.1038/nrg3833.

[2] Sanjay M. Prakadan, Alex K. Shalek, and David A. Weitz. Scaling by shrinking: empowering single-cell 'omics' with microfluidic devices. *Nature Reviews Genetics*, 18(6):345–361, 2017. ISSN 1471-0056. doi: 10.1038/nrg.2017.15. URL http://www.nature.com/doifinder/10.1038/nrg.2017.15.

[3] Simona Patange, Michelle Girvan, and Daniel R. Larson. Single-cell systems biology: Probing the basic unit of information flow. *Current Opinion in Systems Biology*, 8:7–15, 2018. ISSN 24523100. doi: 10.1016/j.coisb.2017.11.011. URL https://doi.org/10.1016/j.coisb.2017.11.011.

[4] Vladimir Yu Kiselev, Tallulah S. Andrews, and Martin Hemberg. Challenges in unsupervised clustering of single-cell RNA-seq data. *Nature Reviews Genetics 2018*, 2019. ISSN 1471-0064. doi: 10.1038/s41576-018-0088-9. URL https://www.nature.com/articles/s41576-018-0088-9?utm{_}source=feedburner{&}utm{_}medium=feed{&}utm{_}campaign=Feed{%}3A+nrg{%}2Frss{%}2Fcurrent+{%}28Nature+Reviews+Genetics+-+Issue{%}29.

[5] Wouter Saelens, Robrecht Cannoodt, Helena Todorov, and Yvan Saeys. A comparison of single-cell trajectory inference methods. *Nature Biotechnology*, 37(5):547–554, May 2019. ISSN 1087-0156, 1546-1696. doi: 10.1038/s41587-019-0071-9.

[6] Mitra Mojtahedi, Alexander Skupin, Joseph Zhou, Ivan G. Castaño, Rebecca Y. Y. Leong-Quong, Hannah Chang, Kalliopi Trachana, Alessandro Giuliani, and Sui Huang. Cell Fate Decision as High-Dimensional Critical State Transition. *PLOS Biology*, 14(12):e2000640, December 2016. ISSN 1545-7885. doi: 10.1371/journal.pbio.2000640.

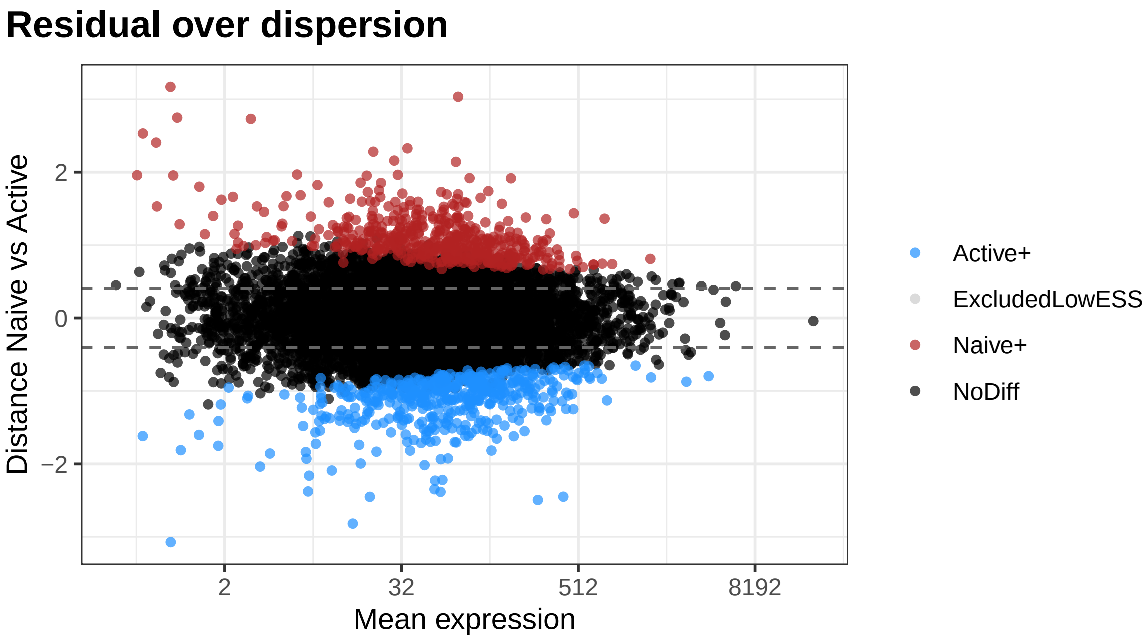

Figure 11. Distance of residual over-dispersion in naive cells relative to active cells. Colour indicates whether genes were excluded from testing, and whether a significant difference was identified between the two groups for that gene.

[7] Celia P. Martinez-Jimenez, Nils Eling, Hung-Chang Chen, Catalina A Vallejos, Aleksandra A Kolodziejczyk, Frances Connor, Lovorka Stojic, Timothy F Rayner, Michael J T Stubbington, Sarah A Teichmann, Maike de la Roche, John C Marioni, and Duncan T Odom. Aging increases cell-to-cell transcriptional variability upon immune stimulation. *Science*, 1436:1433–1436, 2017. doi: 10.1126/science.aah4115.

[8] Michael B Elowitz, Arnold J Levine, Eric D Siggia, and Peter S Swain. Stochastic gene expression in a single cell. *Science*, 297(5584):1183–1186, 2002. ISSN 1095-9203. doi: 10.1126/science.1070919. URL http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve{&}db=PubMed{&}dopt=Citation{&}list{_}uids=12183631.

[9] Nils Eling, Michael D. Morgan, and John C. Marioni. Challenges in measuring and understanding biological noise. *Nature Reviews Genetics*, 20(9):536–548, September 2019. ISSN 1471-0056, 1471-0064. doi: 10.1038/s41576-019-0130-6.

[10] Cristopher J. Zopf, Katie Quinn, Joshua Zeidman, and Narendra Maheshri. Cell-Cycle Dependence of Transcription Dominates Noise in Gene Expression. *PLoS Computational Biology*, 9(7):1–12, 2013. ISSN 1553734X. doi: 10.1371/journal.pcbi.1003161.

[11] Kazunari Iwamoto, Yuki Shindo, and Koichi Takahashi. Modeling Cellular Noise Underlying Heterogeneous Cell Responses in the Epidermal Growth Factor Signaling Pathway. *PLoS Computational Biology*, 12(11):1–18, 2016. ISSN 15537358. doi: 10.1371/journal.pcbi.1005222.

[12] Daniel J. Kiviet, Philippe Nghe, Noreen Walker, Sarah Boulineau, Vanda Sunderlikova, and Sander J. Tans. Stochasticity of metabolism and growth at the single-cell level. *Nature*, 514(7522):376–379, 2014. ISSN 0028-0836. doi: 10.1038/nature13582. URL http://www.nature.com/doifinder/10.1038/nature13582.

[13] James Eberwine and Junhyong Kim. Cellular Deconstruction: Finding Meaning in Individual Cell Variation. *Trends in Cell Biology*, 25(10):569–578, 2015. ISSN 18793088. doi: 10.1016/j.tcb.2015.07.004. URL http://dx.doi.org/10.1016/j.tcb.2015.07.004.

[14] Andre J. Faure, Jörn M. Schmiedel, and Ben Lehner. Systematic Analysis of the Determinants of Gene Expression Noise in Embryonic Stem Cells. *Cell Systems*, 5(5):471–484, 2017. ISSN 24054720. doi: 10.1016/j.cels.2017.10.003.

[15] Michael D. Morgan and John C. Marioni. CpG island composition differences are a source of gene expression noise indicative of promoter responsiveness. *Genome Biology*, 19(1):1–13, 2018. ISSN 1474760X. doi: 10.1186/s13059-018-1461-x.

[16] Philip Brennecke, Simon Anders, Jong Kyoung Kim, Aleksandra A Kołodziejczyk, Xiuwei Zhang, Valentina Proserpio, Bianka Baying, Vladimir Benes, Sarah a Teichmann, John C Marioni, and Marcus G Heisler. Accounting for technical noise in single-cell RNA-seq experiments., 2013. ISSN 1548-7105. URL http://www.ncbi.nlm.nih.gov/pubmed/24056876.

[17] External RNA Controls Consortium. Proposed methods for testing and selecting the ERCC external RNA controls. *BMC Genomics*, 6(June 2004):150, 2005. ISSN 1471-2164. doi: 10.1186/1471-2164-6-150.

[18] Davis J. McCarthy, Kieran R. Campbell, Aaron T.L. Lun, and Quin F. Wills. Scater: Pre-processing, quality control, normalization and visualization of single-cell RNA-seq data in R. *Bioinformatics*, 33(8):1179–1186, 2017. ISSN 14602059. doi: 10.1093/bioinformatics/btw777.

[19] Catalina A Vallejos, Davide Risso, Antonio Scialdone, Sandrine Dudoit, and John C Marioni. Normalizing single-cell RNA sequencing data: challenges and opportunities. *Nature Methods*, 14(6):565–571, 2017. ISSN 1548-7091. doi: 10.1038/nmeth.4292. URL http://www.nature.com/doifinder/10.1038/nmeth.4292.

[20] Saiful Islam, Amit Zeisel, Simon Joost, Gioele La Manno, Pawel Zajac, Maria Kasper, Peter Lönnerberg, and Sten Linnarsson. Quantitative single-cell RNA-seq with unique molecular identifiers. *Nature Methods*, 11(2):163–166, February 2014. ISSN 1548-7091, 1548-7105. doi: 10.1038/nmeth.2772.

[21] Ashraful Haque, Jessica Engel, Sarah A. Teichmann, and Tapio Lönnberg. A practical guide to single-cell RNA-sequencing for biomedical research and clinical applications. *Genome Medicine*, 9(1):75, December 2017. ISSN 1756-994X. doi: 10.1186/s13073-017-0467-4.

[22] Catalina A. Vallejos, John C. Marioni, and Sylvia Richardson. BASiCS: Bayesian analysis of single-cell sequencing data. *PLOS Computational Biology*, 11:e1004333, 2015. ISSN 1553-7358. doi: 10.1371/journal.pcbi.1004333. URL http://dx.plos.org/10.1371/journal.pcbi.1004333.

[23] Catalina A Vallejos, Sylvia Richardson, and John C Marioni. Beyond comparisons of means: understanding changes in gene expression at the single-cell level. *Genome Biology*, 17(70), 2016. doi: 10.1101/035949. URL http://biorxiv.org/content/early/2016/01/05/035949.abstract.

[24] Nils Eling, Arianne C. Richard, Sylvia Richardson, John C. Marioni, and Catalina A. Vallejos. Robust expression variability testing reveals heterogeneous T cell responses. *bioRxiv*, page 237214, 2017. doi: 10.1101/237214. URL https://www.biorxiv.org/content/early/2017/12/21/237214.full.pdf+html.

[25] Valentine Svensson. Droplet scRNA-seq is not zero-inflated. *Nature Biotechnology*, 38(2):147–150, February 2020. ISSN 1087-0156, 1546-1696. doi: 10.1038/s41587-019-0379-5.

[26] F. William Townes. Review of Probability Distributions for Modeling Count Data. *arXiv:2001.04343 [stat]*, January 2020.

[27] F. William Townes, Stephanie C. Hicks, Martin J. Aryee, and Rafael A. Irizarry. Feature selection and dimension reduction for single-cell RNA-Seq based on a multinomial model. *Genome Biology*, 20(1):295, December 2019. ISSN 1474-760X. doi: 10.1186/s13059-019-1861-6.

[28] Nils Eling, Arianne C. Richard, Sylvia Richardson, John C. Marioni, and Catalina A. Vallejos. Correcting the Mean-Variance Dependency for Differential Variability Testing Using Single-Cell RNA Sequencing Data. *Cell Systems*, 7(3): 1–11, 2018. ISSN 24054712. doi: 10.1016/j.cels.2018.06.011. URL https://linkinghub.elsevier.com/retrieve/pii/S2405471218302783.

[29] Aaron T. L. Lun, Davis J. McCarthy, and John C. Marioni. A step-by-step workflow for basic analyses of single-cell RNA-seq data. *F1000Research*, 5(2122), 2016. ISSN 2046-1402. doi: 10.12688/f1000research.9501.1.

[30] Beomseok Kim, Eunmin Lee, and Jong Kyoung Kim. Analysis of Technical and Biological Variabilityin Single-Cell RNA Sequencing. In *Computational Methods for Single-Cell Data Analysis*, volume 1935, pages 25–43. 2019. ISBN 978-1-4939-9056-6. doi: 10.1007/978-1-4939-9057-3. URL http://www.ncbi.nlm.nih.gov/pubmed/30758827{%}0Ahttp://link.springer.com/10.1007/978-1-4939-9057-3{_}12{%}0Ahttp://link.springer.com/10.1007/978-1-4939-9057-3.

[31] Carl Boettiger. An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1): 71–79, January 2015. ISSN 0163-5980. doi: 10.1145/2723872.2723882.

[32] Robert A. Amezquita, Vince J. Carey, Lindsay N. Carpp, Ludwig Geistlinger, Aaron T. L. Lun, Federico Marini, Kevin Rue-Albrecht, Davide Risso, Charlotte Soneson, Levi Waldron, Hervé Pagès, Mike Smith, Wolfgang Huber, Martin Morgan, Raphael Gottardo, and Stephanie C. Hicks. Orchestrating Single-Cell Analysis with Bioconductor. Preprint, Genomics, March 2019.

[33] Tomislav Ilicic, Jong Kyoung Kim, Aleksandra A Kolodziejczyk, Frederik Otzen Bagger, Davis James Mccarthy, John C Marioni, and Sarah A Teichmann. Classification of low quality cells from single-cell RNA-seq data. *Genome Biology*, 17(29):1–15, 2016. ISSN 1474-760X. doi: 10.1186/s13059-016-0888-1. URL http://dx.doi.org/10.1186/s13059-016-0888-1.

[34] Aaron T. L. Lun, Karsten Bach, and John C. Marioni. Pooling across cells to normalize single-cell RNA sequencing data with many zero counts. *Genome Biology*, 17(1):75, 2016. ISSN 1474-760X. doi: 10.1186/s13059-016-0947-7. URL http://genomebiology.biomedcentral.com/articles/10.1186/s13059-016-0947-7.

[35] Aleksandra A. Kolodziejczyk, Jong Kyoung Kim, Jason C.H. Tsang, Tomislav Ilicic, Johan Henriksson, Kedar N. Natarajan, Alex C. Tuck, Xuefei Gao, Marc Bühler, Pentao Liu, John C. Marioni, and Sarah A. Teichmann. Single cell RNA-sequencing of pluripotent states unlocks modular transcriptional variation. *Cell Stem Cell*, 17:471–485, 2015. ISSN 19345909. doi: 10.1016/j.stem.2015.09.011. URL http://linkinghub.elsevier.com/retrieve/pii/S193459091500418X.

[36] Catalina A. Vallejos, John C. Marioni, and Sylvia Richardson. BASiCS: Bayesian analysis of single-cell sequencing data. *PLOS Computational Biology*, 11(6):e1004333, 2015. ISSN 1553-7358. doi: 10.1371/journal.pcbi.1004333. URL http://dx.plos.org/10.1371/journal.pcbi.1004333.

[37] Yingxin Lin, Shila Ghazanfar, Dario Strbenac, Andy Wang, Ellis Patrick, David M Lin, Terence Speed, Jean Y H Yang, and Pengyi Yang. Evaluating stably expressed genes in single cells. *GigaScience*, 8(9):giz106, September 2019. ISSN 2047-217X. doi: 10.1093/gigascience/giz106.

[38] Peter V Kharchenko, Lev Silberstein, and David T Scadden. Bayesian approach to single-cell differential expression analysis. *Nature methods*, 11(7):740–2, jul 2014. ISSN 1548-7105. doi: 10.1038/nmeth.2967. URL http://www.ncbi.nlm.nih.gov/pubmed/24836921.

[39] Greg Finak, Andrew McDavid, Masanao Yajima, Jingyuan Deng, Vivian Gersuk, Alex K. Shalek, Chloe K. Slichter, Hannah W. Miller, M. Juliana McElrath, Martin Prlic, Peter S. Linsley, and Raphael Gottardo. MAST: A flexible statistical framework for assessing transcriptional changes and characterizing heterogeneity in single-cell RNA sequencing data. *Genome Biology*, 16(1):278, December 2015. ISSN 1474-760X. doi: 10.1186/s13059-015-0844-5.

[40] Charlotte Soneson and Mark D Robinson. Bias, robustness and scalability in single-cell differential expression analysis. *Nature Methods*, 15(4):255–261, April 2018. ISSN 1548-7091, 1548-7105. doi: 10.1038/nmeth.4612.

[41] David Lähnemann, Johannes Köster, Ewa Szczurek, Davis J. McCarthy, Stephanie C. Hicks, Mark D. Robinson, Catalina A. Vallejos, Kieran R. Campbell, Niko Beerenwinkel, Ahmed Mahfouz, Luca Pinello, Pavel Skums, Alexandros Stamatakis, Camille Stephan-Otto Attolini, Samuel Aparicio, Jasmijn Baaijens, Marleen Balvert, Buys de Barbanson, Antonio Cappuccio, Giacomo Corleone, Bas E. Dutilh, Maria Florescu, Victor Guryev, Rens Holmer, Katharina Jahn, Thamar Jessurun Lobo, Emma M. Keizer, Indu Khatri, Szymon M. Kielbasa, Jan O. Korbel, Alexey M. Kozlov, Tzu-Hao Kuo, Boudewijn P.F. Lelieveldt, Ion I. Mandoiu, John C. Marioni, Tobias Marschall, Felix Mölder, Amir Niknejad, Lukasz Raczkowski, Marcel Reinders, Jeroen de Ridder, Antoine-Emmanuel Saliba, Antonios Somarakis, Oliver Stegle, Fabian J. Theis, Huan Yang, Alex Zelikovsky, Alice C. McHardy, Benjamin J. Raphael, Sohrab P. Shah, and Alexander Schönhuth. Eleven grand challenges in single-cell data science. *Genome Biology*, 21(1):31, December 2020. ISSN 1474-760X. doi: 10.1186/s13059-020-1926-6.

[42] Steffen Durinck, Paul T Spellman, Ewan Birney, and Wolfgang Huber. Mapping identifiers for the integration of genomic datasets with the R/Bioconductor package biomaRt. *Nature Protocols*, 4(8):1184–1191, August 2009. ISSN 1754-2189, 1750-2799. doi: 10.1038/nprot.2009.97.

[43] Romain Lopez, Jeffrey Regier, Michael B. Cole, Michael I. Jordan, and Nir Yosef. Deep generative modeling for single-cell transcriptomics. *Nature Methods*, 15(12):1053–1058, December 2018. ISSN 1548-7091, 1548-7105. doi: 10.1038/s41592-018-0229-2.

[44] Gareth O. Roberts and Jeffrey S. Rosenthal. Examples of Adaptive MCMC. *Journal of Computational and Graphical Statistics*, 18(2):349–367, January 2009. ISSN 1061-8600, 1537-2715. doi: 10.1198/jcgs.2009.06134.

[45] Mary Kathryn Cowles and Bradley P. Carlin. Markov chain monte carlo convergence diagnostics: A comparative review. *Journal of the American Statistical Association*, 91(434):883–904, 1996. doi: 10.1080/01621459.1996.10476956. URL https://amstat.tandfonline.com/doi/abs/10.1080/01621459.1996.10476956.

[46] Stephen P. Brooks and Andrew Gelman. General methods for monitoring convergence of iterative simulations. *Journal of Computational and Graphical Statistics*, 7(4):434–455, 1998. doi: 10.1080/10618600.1998.10474787. URL https://amstat.tandfonline.com/doi/abs/10.1080/10618600.1998.10474787.

[47] John Geweke and Forthcoming In. Evaluating the accuracy of sampling-based approaches to the calculation of posterior moments. 4, 11 1995.

[48] Elizabeth Koehler, Elizabeth Brown, and Sebastien J.-P. A. Haneuse. On the Assessment of Monte Carlo Error in Simulation-Based Statistical Analyses. *The American Statistician*, 63(2):155–162, May 2009. ISSN 0003-1305, 1537-2731. doi: 10.1198/tast.2009.0030.

[49] Andrew Gelman, John Carlin, Hal Stern, David B Dunson, Aki Vehtari, and Donald B Rubin. *Bayesian Data Analysis*. CRC Press, 2014. ISBN 978-1439840955.

[50] Vlatka Antolović, Agnes Miermont, Adam M. Corrigan, and Jonathan R. Chubb. Generation of Single-Cell Transcript Variability by Repression. *Current Biology*, 27:1811–1817, 2017. ISSN 09609822. doi: 10.1016/j.cub.2017.05.028. URL http://linkinghub.elsevier.com/retrieve/pii/S096098221730564X.

[51] M. A. Newton. Detecting differential gene expression with a semiparametric hierarchical mixture method. *Biostatistics*, 5(2):155–176, April 2004. ISSN 1465-4644, 1468-4357. doi: 10.1093/biostatistics/5.2.155.

[52] J Adam Best, Jamie Knell, Edward Yang, Viveka Mayya, Andrew Doedens, Michael L Dustin, and Ananda W Goldrath. Transcriptional insights into the CD8+ T cell response to infection and memory T cell formation. *Nature Immunology*, 14(4):404–412, April 2013. ISSN 1529-2908, 1529-2916. doi: 10.1038/ni.2536.

[53] Wenxian Fu, Ayla Ergun, Ting Lu, Jonathan A Hill, Sokol Haxhinasto, Marlys S Fassett, Roi Gazit, Stanley Adoro, Laurie Glimcher, Susan Chan, Philippe Kastner, Derrick Rossi, James J Collins, Diane Mathis, and Christophe Benoist. A multiply redundant genetic switch 'locks in' the transcriptional signature of regulatory T cells. *Nature Immunology*, 13(10):972–980, October 2012. ISSN 1529-2908, 1529-2916. doi: 10.1038/ni.2420.

[54] Jinfang Zhu and William E. Paul. Peripheral CD4+ T-cell differentiation regulated by networks of cytokines and transcription factors: Transcription factor network in Th cells. *Immunological Reviews*, 238(1):247–262, November 2010. ISSN 01052896. doi: 10.1111/j.1600-065X.2010.00951.x.

[55] Matthew D Young, Matthew J Wakefield, Gordon K Smyth, and Alicia Oshlack. Gene ontology analysis for RNA-seq: accounting for selection bias. *Genome Biology*, 11, 2010. URL http://genomebiology.com/2010/11/2/R14.

[56] Steven F. Ziegler, Fred Ramsdell, and Mark R. Alderson. The activation antigen CD69. *Stem Cells*, 12(5):456–465, 1994. ISSN 10665099, 15494918. doi: 10.1002/stem.5530120502.

[57] Zuguang Gu, Roland Eils, and Matthias Schlesner. Complex heatmaps reveal patterns and correlations in multidimensional genomic data. *Bioinformatics*, 2016.