

Bioconductor workflow for single-cell RNA sequencing: Normalization, dimensionality reduction, clustering, and lineage inference

Fanny Perraudeau¹, Davide Risso², Kelly Street¹, Elizabeth Purdom³, and Sandrine Dudoit⁴

¹Graduate Group in Biostatistics, University of California, Berkeley, Berkeley, CA

²Division of Biostatistics and Epidemiology, Department of Healthcare Policy and Research, Weill Cornell Medicine, New York, NY

³Department of Statistics, University of California, Berkeley, Berkeley, CA

⁴Division of Biostatistics and Department of Statistics, University of California, Berkeley, Berkeley, CA

Abstract Novel single-cell transcriptome sequencing assays allow researchers to measure gene expression levels at the resolution of single cells and offer the unprecedented opportunity to investigate at the molecular level fundamental biological questions such as stem cell differentiation or the discovery and characterization of rare cell types. However, such assays raise challenging statistical and computational questions and require the development of novel methodology and software. Using stem cell differentiation in the mouse olfactory epithelium as a case study, this integrated workflow provides a step-by-step tutorial to the methodology and associated software for the following four main tasks:(1) dimensionality reduction accounting for zero inflation and over-dispersion and adjusting for gene and cell-level covariates; (2) cell clustering using resampling-based sequential ensemble clustering; (3) inference of cell lineages and pseudotimes; and (4) differential expression analysis along lineages.

Keywords

single-cell, RNA-seq, normalization, dimensionality reduction, clustering, lineage inference, differential expression, workflow

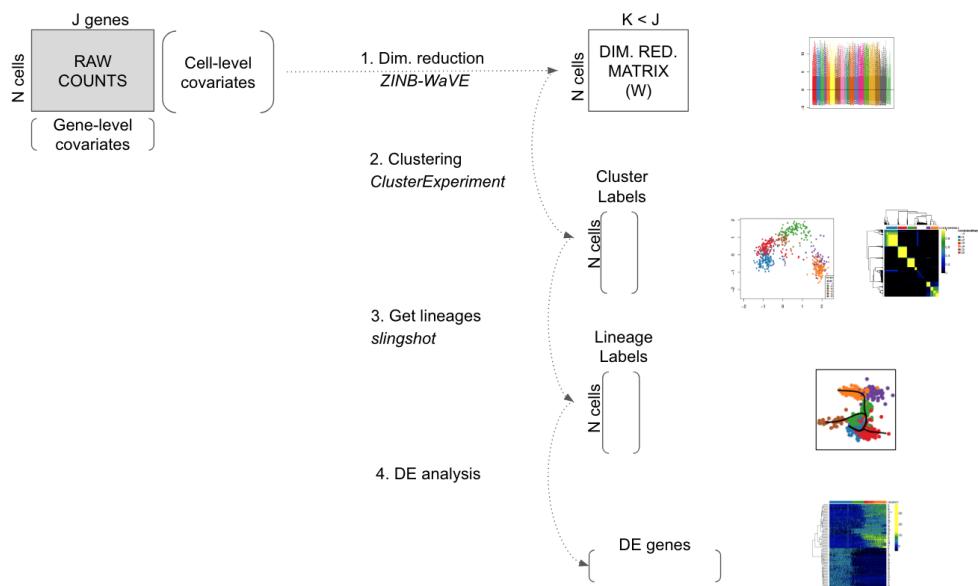


Figure 1. Workflow for analyzing scRNA-seq datasets. On the right, main plots generated by the workflow.

Introduction

Single-cell RNA sequencing (scRNA-seq) is a powerful and promising class of high-throughput assays that enable researchers to measure genome-wide transcription levels at the resolution of single cells. To properly account for features specific to scRNA-seq, such as zero inflation and high levels of technical noise, several novel statistical methods have been developed to tackle questions that include normalization, dimensionality reduction, clustering, the inference of cell lineages and pseudotime, and the identification of differentially expressed (DE) genes. While each individual method is useful on its own for addressing a specific question, there is an increasing need for workflows that integrate these tools to yield a seamless scRNA-seq data analysis pipeline. This is all the more true, with novel sequencing technologies that allow an increasing number of cells to be sequenced in each run. For example, the Chromium Single Cell 3' Solution was recently used to sequence and profile about 1.3 million cells from embryonic mouse brains.

scRNA-seq low-level analysis workflows have already been developed, with useful methods for quality control (QC), exploratory data analysis (EDA), pre-processing, normalization, and visualization. The workflow described in [1] and the package `scater` [2] are such examples based on open-source R software packages from the Bioconductor Project [3]. In these workflows, single-cell expression data are organized in objects of the `SingleCellExperiment` class allowing integrated analysis [4]. However, these workflows are mostly used to prepare the data for further downstream analysis and do not focus on steps such as cell clustering and lineage inference.

Here, we propose an integrated workflow for downstream analysis, with the following four main steps: (1) dimensionality reduction accounting for zero inflation and over-dispersion and adjusting for gene and cell-level covariates, using the `zinbwave` Bioconductor package; (2) robust and stable cell clustering using resampling-based sequential ensemble clustering, as implemented in the `clusterExperiment` Bioconductor package; (3) inference of cell lineages and ordering of the cells by developmental progression along lineages, using the `slingshot` Bioconductor package; and (4) DE analysis along lineages.

One of the cornerstones of Bioconductor is the use of common data structures, implemented as classes in the S4 object-oriented system of R [5]. Throughout the workflow, we use a single object of the `SingleCellExperiment` class to store the scRNA-seq data along with any gene or cell-level metadata available from the experiment. The `SingleCellExperiment` class [4] is an extension of the `SummarizedExperiment` class [6] specifically designed for single-cell data (see Figure 2 of [5] for an overview of `SummarizedExperiment`). The use of the `SingleCellExperiment` class allows us to seamlessly transition between the steps of this workflow, as well as to integrate this workflow with the upstream QC and EDA steps provided in [1].

Analysis of olfactory stem cell differentiation using scRNA-seq data

Overview

This workflow is illustrated using data from a scRNA-seq study of stem cell differentiation in the mouse olfactory epithelium (OE) [7]. The olfactory epithelium contains mature olfactory sensory neurons (mOSN) that

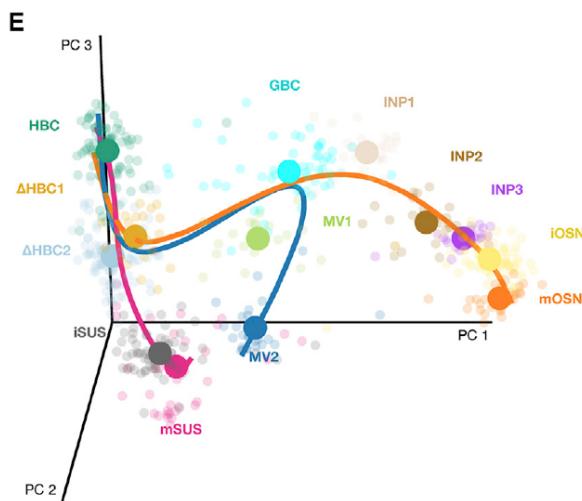


Figure 2. Stem cell differentiation in the mouse olfactory epithelium. This figure was reproduced with kind permission from Fletcher et al. (2017).

are continuously renewed in the epithelium via neurogenesis through the differentiation of globose basal cells (GBC), which are the actively proliferating cells in the epithelium. When a severe injury to the entire tissue happens, the olfactory epithelium can regenerate from normally quiescent stem cells called horizontal basal cells (HBC), which become activated to differentiate and reconstitute all major cell types in the epithelium.

The scRNA-seq dataset we use as a case study was generated to study the differentiation of HBC stem cells into different cell types present in the olfactory epithelium. To map the developmental trajectories of the multiple cell lineages arising from HBCs, scRNA-seq was performed on FACS-purified cells using the Fluidigm C1 microfluidics cell capture platform followed by Illumina sequencing. The expression level of each gene in a given cell was quantified by counting the total number of reads mapping to it. Cells were then assigned to different lineages using a statistical analysis pipeline analogous to that in the present workflow. Finally, results were validated experimentally using *in vivo* lineage tracing. Details on data generation and statistical methods are available in [7, 8, 9, 10].

It was found that the first bifurcation produces two lineage trajectories: the first one (purple curve in Figure 2) generates mature sustentacular cells (mSUS) through a differentiation process characterized by the absence of cell cycle and division. The other lineage trajectory (blue and orange curves in Figure 2) generates proliferative GBCs, which are characterized by cell cycle and division. These, in turn, generate mature olfactory sensory neurons (mOSN; orange curve) and microvillous cells (MV; blue curve).

In this workflow, we describe a sequence of steps to recover the lineages found in the original study, starting from the genes x cells matrix of raw counts publicly-available at <https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE95601>.

Note that the preprocessing steps needed to produce such a count matrix from the raw FASTQ files are outside of the scope of this paper. These are very important steps that should be taken with care and the optimal workflow depends on the technology and the protocol used to collect the data (e.g., 3' end sequencing, droplet-based methods versus full-length mRNA, plate-based methods). One way to carry out these steps within the Bioconductor framework is the `scPipe` package [11], which provides functions to quantify gene expression starting from raw sequencing files. More details can be found in the `scPipe` vignette, available at <https://bioconductor.org/packages/scPipe>.

Package installation and versions

The source code to reproduce the analysis in the workflow are available at <https://github.com/fperraudreau/singlecellworkflow>. The packages used in the workflow can be installed using the following in R:

```
install.packages(c("doParallel", "gam", "RColorBrewer"))

install.packages("BiocManager")

BiocManager::install(c("scone", "zinbwave", "clusterExperiment", "slingshot",
                      "BiocParallel"))
```

Note that in order to successfully run the workflow, we need the following versions of the Bioconductor packages `scone` ($>=1.5.0$), `zinbwave` ($>=1.3.2$), and `clusterExperiment` ($>=2.1.3$). We recommend running Bioconductor 3.8 (currently the devel version; see <https://www.bioconductor.org/developers/how-to/useDevel/>).

We are now ready to load the packages.

```
# Bioconductor
library(BiocParallel)
library(clusterExperiment)
library(scone)
library(zinbwave)
library(slingshot)

# CRAN
library(doParallel)
library(gam)
library(RColorBrewer)

set.seed(20)
```

Parallel computing

To give an idea to the users of the time needed to run the workflow, function `system.time` was used to report computation times for the time consuming functions. Computations were performed with 6 cores on a iMac with a 4 GHz Intel Core i7 processor and 32 GB of RAM. The `BiocParallel` package is used to allow for parallel computing in the `zinbwave` function. Users with a different operating system may change the package used for parallel computing and the `NCORES` variable below.

```
NCORES <- 6
mysystem = Sys.info() [["sysname"]]
if (mysystem == "Darwin"){
  registerDoParallel(NCORES)
  register(DoparParam())
} else if (mysystem == "Linux"){
  register(bpstart(MulticoreParam(workers=NCORES)))
} else{
  print("Please change this to allow parallel computing on your computer.")
  register(SerialParam())
}
```

Pre-processing

Counts for all genes in each cell were obtained from NCBI Gene Expression Omnibus (GEO), with accession number GSE95601. Before filtering, the dataset has 849 cells and 28,361 detected genes (i.e., genes with non-zero read counts).

Note that in the following, we assume that the user has access to a data folder located at `../data`. Users with a different directory structure may need to change the `data_dir` variable below to reproduce the workflow.

```
data_dir <- "../data/"

urls = c("https://www.ncbi.nlm.nih.gov/geo/download/?acc=GSE95601&format=file&file=GSE95601%5FoeHBCdiff_eSet.Rda",
        "https://raw.githubusercontent.com/rufletch/p63-HBC-diff/master/ref/oeHBCdiff_clusterLabels.txt")

if(!file.exists(paste0(data_dir, "GSE95601_oeHBCdiff_Cufflinks_eSet.Rda"))){
  download.file(urls[1], paste0(data_dir, "GSE95601_oeHBCdiff_Cufflinks_eSet.Rda.gz"))
  R.utils::gunzip(paste0(data_dir, "GSE95601_oeHBCdiff_Cufflinks_eSet.Rda.gz"))
}

if(!file.exists(paste0(data_dir, "oeHBCdiff_clusterLabels.txt"))){
  download.file(urls[2], paste0(data_dir, "oeHBCdiff_clusterLabels.txt"))
}
```

```

load(paste0(data_dir, "GSE95601_oeHBCdiff_Cufflinks_eSet.Rda"))

# Count matrix
E <- assayData(Cufflinks_eSet)$counts_table

# Remove undetected genes
E <- na.omit(E)
E <- E[rowSums(E)>0,]
dim(E)

## [1] 28361 849

```

We remove the ERCC spike-in sequences and the CreER gene, as the latter corresponds to the estrogen receptor fused to Cre recombinase (Cre-ER), which is used to activate HBCs into differentiation following injection of tamoxifen (see [7] for details).

DR: Consider keeping the spike-ins in the object to show the ability of SCE to use spike-ins... but we don't use them, so not sure how important it is.

```

# Remove ERCC and CreER genes
cre <- E["CreER",]
ercc <- E[grep("ERCC-", rownames(E)),]
E <- E[grep("ERCC-", rownames(E), invert = TRUE), ]
E <- E[-which(rownames(E)=="CreER"), ]
dim(E)

```

```
## [1] 28284 849
```

The cell-level metadata contain quality control measures, sequencing batch ID, and cluster and lineage labels from the original publication [7]. Cells with a cluster label of -2 were not assigned to any cluster in the original publication.

```

# Extract QC metrics
qc <- as.matrix(protocolData(Cufflinks_eSet)$data)[,c(1:5, 10:18)]
qc <- cbind(qc, CreER = cre, ERCC_reads = colSums(ercc))

# Extract metadata
batch <- droplevels(pData(Cufflinks_eSet)$MD_c1_run_id)
bio <- droplevels(pData(Cufflinks_eSet)$MD_expt_condition)
clusterLabels <- read.table(paste0(data_dir, "oeHBCdiff_clusterLabels.txt"),
                           sep = "\t", stringsAsFactors = FALSE)
m <- match(colnames(E), clusterLabels[, 1])

# Create metadata data.frame
metadata <- data.frame("Experiment" = bio,
                       "Batch" = batch,
                       "publishedClusters" = clusterLabels[m, 2],
                       qc)

# Symbol for cells not assigned to a lineage in original data
metadata$publishedClusters[is.na(metadata$publishedClusters)] <- -2

```

As discussed above, we store the data in an object of the class `SingleCellExperiment`. In addition to keeping the expression values and their associated metadata within a single object, `SingleCellExperiment` allows the user to store normalization factors and the results of dimensionality reduction methods (such as `zinbwave`) within the object. Throughout the workflow, we will highlight when this special features of `SingleCellExperiment` are used.

```

dataObj <- SingleCellExperiment(assays = list(counts = E),
                                colData = metadata)
dataObj

```

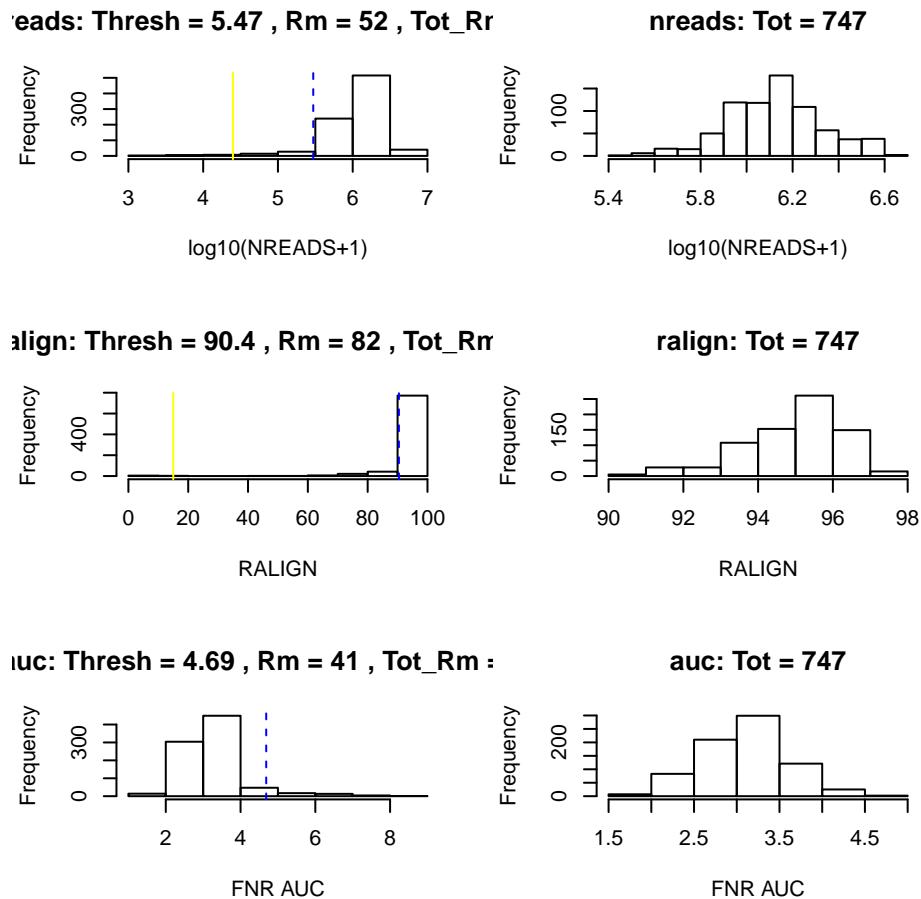


Figure 3. SCONE: Filtering of low-quality cells.

```
## class: SingleCellExperiment
## dim: 28284 849
## metadata(0):
## assays(1): counts
## rownames(28284): Xkr4 LOC102640625 ... Ggcx.1 eGFP
## rowData names(0):
## colnames(849): OEP01_N706_S501 OEP01_N701_S501 ... OEL23_N704_S503
##   OEL23_N703_S502
## colData names(19): Experiment Batch ... CreER ERCC_reads
## reducedDimNames(0):
## spikeNames(0):
```

Using the Bioconductor R package `scone` [12], we remove low-quality cells according to the quality control filter implemented in the function `metric_sample_filter` and based on the following criteria (Figure 3): (1) Filter out samples with low total number of reads or low alignment percentage and (2) filter out samples with a low detection rate for housekeeping genes. See the `scone` vignette for details on the filtering procedure.

```
# QC-metric-based sample-filtering
data("housekeeping")
hk = rownames(dataObj)[toupper(rownames(dataObj)) %in% housekeeping$V1]

mfilt <- metric_sample_filter(counts(dataObj),
                               nreads = colData(dataObj)$NREADS,
                               ralign = colData(dataObj)$RALIGN,
                               pos_controls = rownames(dataObj) %in% hk,
                               zcut = 3, mixture = FALSE,
                               plot = TRUE)

# Simplify to a single logical
mfilt <- !apply(simplify2array(mfilt[!is.na(mfilt)]), 1, any)
```

```
dataObj <- dataObj[, mfilt]
dim(dataObj)
```

```
## [1] 28284    747
```

After sample filtering, we are left with 747 good quality cells.

Identification of most variable genes

Often in unsupervised analyses of expression data, it is useful to focus on the most variable genes. These genes are the ones that vary the most across cells and are likely to be the most informative. Filtering out low-variance genes allows us to reduce the complexity of the problem, while retaining the most information in the dataset.

Here, for computational efficiency, we retain only the 1,000 most variable genes, using a simple $\log(x+1)$ transformation. This seems to be a reasonable choice for the illustrative purpose of this workflow, as we are able to recover the biological signal found in the published analysis ([7]). In general, however, we recommend care in selecting a gene filtering scheme, as an appropriate choice is dataset-dependent and many different strategies have been proposed in the literature.

To compute the gene-wise variances and keep only the most variable genes, we use two functions from the `clusterExperiment` package. In particular, `makeFilterStats` will compute the variance of the gene expression data with a given transformation applied to the counts (in this case $\log(x+1)$). It will then store the variance values in the `rowData` slot of the object. `filterData` will use the computed values to keep only the desired number of genes.

```
dataObj <- makeFilterStats(dataObj, filterStats = "var", transFun = log1p)
dataObj <- filterData(dataObj, percentile=1000, filterStats="var")
rowData(dataObj)
```

```
## DataFrame with 1000 rows and 1 column
##           var
##      <numeric>
## Cbr2     12.0552486323842
## Cyp2f2   11.9232499641982
## Gstm1    11.7632685207579
## Sec14l3  11.3453516639106
## Cyp2g1   10.7622947197034
## ...
## Krit1    4.85641748262887
## Sri      4.85625337610767
## Hdac4    4.85536560153121
## Rnf13    4.85219227946459
## Atp7b    4.85157190635962
```

One drawback of this strategy is that the logarithm is not a variance-stabilizing transformation. Variance stabilizing transformations based on the negative binomial distribution are available, e.g., in the `DESeq2` package. And these may be a better choice in real data analyses. Note that the `makeFilterStats` function of `clusterExperiment` can be used with any user-supplied function. Here, we show an example with the `vst` function from `DESeq2`.

```
library(DESeq2)
dataObj <- makeFilterStats(dataObj, filterStats = "var", transFun = vst,
                           filterNames = "var_vst")
dataObj <- filterData(dataObj, percentile=1000, filterStats="var_vst")
```

Alternatively, the `scran` package [1] includes the `trendVar` and `decomposeVar` functions, which estimate the mean-variance relation and the technical and biological components of the variance, to select those genes with a high biological variance. See [1] for details.

DR: I'm not sure whether to keep this last example using `scran`. It's more complicated because we cannot directly use the `clusterExperiment` functions and it's somewhat involved (we would need to explain what the size factors are etc.). Also, the results make no sense (negative variances!). Perhaps we should just mention the method without the code?

Dataset structure

Overall, after the above pre-processing steps, our dataset has 1,000 genes and 747 cells.

```
dataObj
```

```
## class: SingleCellExperiment
## dim: 1000 747
## metadata(0):
## assays(1): counts
## rownames(1000): Cbr2 Cyp2f2 ... Rnf13 Atp7b
## rowData names(1): var
## colnames(747): OEP01_N706_S501 OEP01_N701_S501 ... OEL23_N704_S503
##   OEL23_N703_S502
## colData names(19): Experiment Batch ... CreER ERCC_reads
## reducedDimNames(0):
## spikeNames(0):
```

Metadata for the cells are stored in the slot `colData` from the `SummarizedExperiment` object. Cells were processed in 18 different runs of the Fluidigm C1 system: we refer to each run as a “batch”, and we refer to the technical differences between each run as “batch effects”. Note that in this particular case, the batch structure is essentially defined by the limitation of the C1 system to process up to 96 cells at a time and it is hence natural to identify each C1 run as one batch. Batch effects are ubiquitous in high-throughput genomics [13] and single-cell RNA-seq is not an exception [14]. In other contexts, depending on experimental design, batches may correspond to different library preparations, different labs in which the data is processed, or different times at which samples were collected.

```
batch <- colData(dataObj)$Batch
col_batch = c(brewer.pal(9, "Set1"), brewer.pal(8, "Dark2"),
             brewer.pal(8, "Accent")[1])
names(col_batch) = unique(batch)
table(batch)
```

```
## batch
## GBC08A GBC08B GBC09A GBC09B      P01      P02      P03A      P03B      P04      P05
##    39     40     35     22      31      48      51      40      20      23
##    P06     P10     P11     P12      P13      P14      Y01      Y04
##    51     40     50     50      60      47      58      42
```

In the original work [7], cells were clustered into 14 different clusters, with 151 cells not assigned to any cluster (i.e., cluster label of -2).

```
publishedClusters <- colData(dataObj)[, "publishedClusters"]
col_clus <- c("transparent", "#1B9E77", "antiquewhite2", "cyan", "#E7298A",
             "#A6CEE3", "#666666", "#E6AB02", "#FFED6F", "darkorchid2",
             "#B3DE69", "#FF7F00", "#A6761D", "#1F78B4")
names(col_clus) <- sort(unique(publishedClusters))
table(publishedClusters)

## publishedClusters
## -2  1  2  3  4  5  7  8  9  10 11 12 14 15
## 151 90 25 54 35 93 58 27 74 26 21 35 26 32
```

Note that there is partial nesting of batches within clusters (i.e., cell type), which could be problematic when correcting for batch effects in the dimensionality reduction step below.

```
table(data.frame(batch = as.vector(batch),
                 cluster = publishedClusters))
```

```
##           cluster
## batch      -2  1  2  3  4  5  7  8  9 10 11 12 14 15
##   GBC08A    3  0  2 12  9  0  0  0  0  2  0  2  9
##   GBC08B    8  0  7  5  3  0  0  0  1  2  3  0  5  6
```

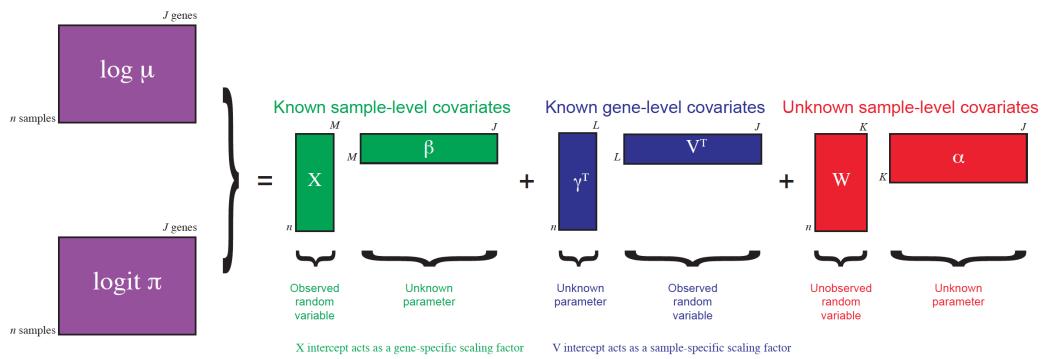


Figure 4. ZINB-WaVE: Schematic view of the ZINB-WaVE model. This figure was reproduced with kind permission from Risso et al. (2017).

```
##   GBC09A 6 0 1 5 8 0 0 0 1 1 0 0 0 6 7
##   GBC09B 12 0 2 1 3 0 0 0 1 0 0 0 0 3 0
##   P01    7 0 2 4 3 15 0 0 0 0 0 0 0 0 0 0
##   P02    5 2 0 9 3 15 3 3 2 3 0 2 1 0 0 0
##   P03A   15 3 0 2 0 12 2 9 4 2 0 2 0 0 0 0
##   P03B   9 1 2 1 1 11 1 2 8 1 1 2 0 0 0 0
##   P04    8 0 0 0 0 9 1 0 1 1 0 0 0 0 0 0 0
##   P05    3 0 0 0 1 11 3 0 1 0 2 2 0 0 0 0
##   P06    12 1 2 3 0 8 2 4 8 4 1 2 2 2 2 2
##   P10    7 3 1 4 0 3 5 8 1 0 2 5 0 1 1 1
##   P11    6 2 1 1 0 1 5 1 22 3 1 6 0 1 1 1
##   P12    10 0 2 0 0 4 10 0 8 2 3 6 4 1 1 1
##   P13    13 1 2 4 0 4 15 0 4 5 6 1 3 2 1 1
##   P14    9 0 0 1 2 0 11 0 12 2 0 7 0 3 1 1
##   Y01    8 46 1 1 2 0 0 0 0 0 0 0 0 0 0 0 0
##   Y04    10 31 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Normalization and dimensionality reduction: ZINB-WaVE

In scRNA-seq analysis, dimensionality reduction is often used as a preliminary step prior to downstream analyses, such as clustering, cell lineage and pseudotime ordering, and the identification of DE genes. This allows the data to become more tractable, both from a statistical (cf. curse of dimensionality) and computational point of view. Additionally, technical noise can be reduced while preserving the often intrinsically low-dimensional signal of interest [15, 16, 8].

Here, we perform dimensionality reduction using the zero-inflated negative binomial-based wanted variation extraction (ZINB-WaVE) method implemented in the Bioconductor R package `zinbwave`. The method fits a ZINB model that accounts for zero inflation (dropouts), over-dispersion, and the count nature of the data. The model can include a cell-level intercept, which serves as a global-scaling normalization factor. The user can also specify both gene-level and cell-level covariates. The inclusion of observed and unobserved cell-level covariates enables normalization for complex, non-linear effects (often referred to as batch effects), while gene-level covariates may be used to adjust for sequence composition effects (e.g., gene length and GC-content effects). A schematic view of the ZINB-WaVE model is provided in Figure 4. For greater detail about the ZINB-WaVE model and estimation procedure, please refer to the original manuscript [8].

As with most dimensionality reduction methods, the user needs to specify the number of dimensions for the new low-dimensional space. Here, we use $K = 50$ dimensions and adjust for batch effects via the matrix X . Note that if the users include more genes in the analysis, it may be preferable to reduce K to achieve a similar computational time.

```
print(system.time(dataObj <- zinbwave(dataObj, K = 10, X = "~- Batch",
                                         residuals = TRUE,
                                         normalizedValues = TRUE)))
```

```
##   user  system elapsed
## 635.061 145.164 163.341
```

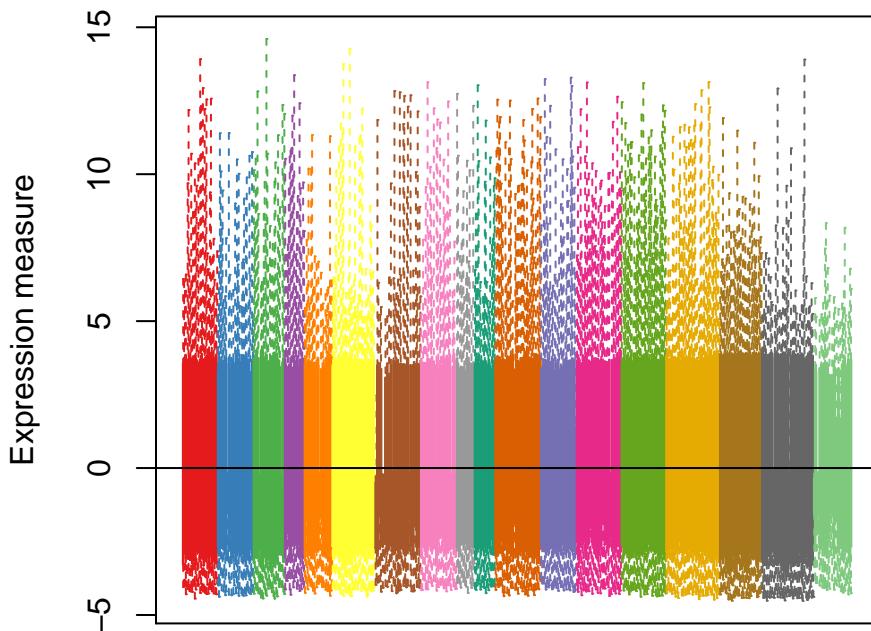


Figure 5. ZINB-WaVE: Boxplots of normalized expression measures (deviance residuals), color-coded by batch.

```
save(dataObj, file= 'dataObj_afterZinbwave_k10.rda')
#load('dataObj_afterZinbwave.rda')
```

Normalization

The function `zinbwave` returns a `SingleCellExperiment` object that includes the normalized expression measures (`normalizedValues`), defined as deviance residuals from the fit of the ZINB-WaVE model with user-specified gene- and cell-level covariates. Such residuals can be used for visualization purposes (e.g., in heatmaps, boxplots). Note that, in this case, the low-dimensional matrix W is not included in the computation of residuals to avoid the removal of the biological signal of interest.

```
norm <- assays(dataObj)$normalizedValues
norm[1:3,1:3]

##          OEP01_N706_S501 OEP01_N701_S501 OEP01_N707_S507
## Cbr2        4.560644     4.406066    -4.195526
## Cyp2f2      4.346453     4.297935     4.127072
## Gstm1       4.723611     4.607802     4.416433
```

As expected, the normalized values no longer exhibit batch effects (Figure 5).

```
norm_order <- norm[, order(as.numeric(batch))]
col_order <- col_batch[batch[order(as.numeric(batch))]]
boxplot(norm_order, col = col_order, staplewex = 0, outline = 0,
        border = col_order, xaxt = "n", ylab="Expression measure")
abline(h=0)
```

We are going to set these normalized values to be our first assay, so that they become the default in future plots.

```
assays(dataObj)<-assays(dataObj)[c("normalizedValues", "residuals", "counts", "weights")]
```

In addition to the normalized values, the `zinbwave` function added several new components to the object: the assay `residuals` contains the residuals of the model that can be used to check the goodness-of-fit of the model. The `weights` assay contains observational weights that can be used to downweight the effect of zero counts on differential expression (see [17] and the `zinbwave` vignette for more details).

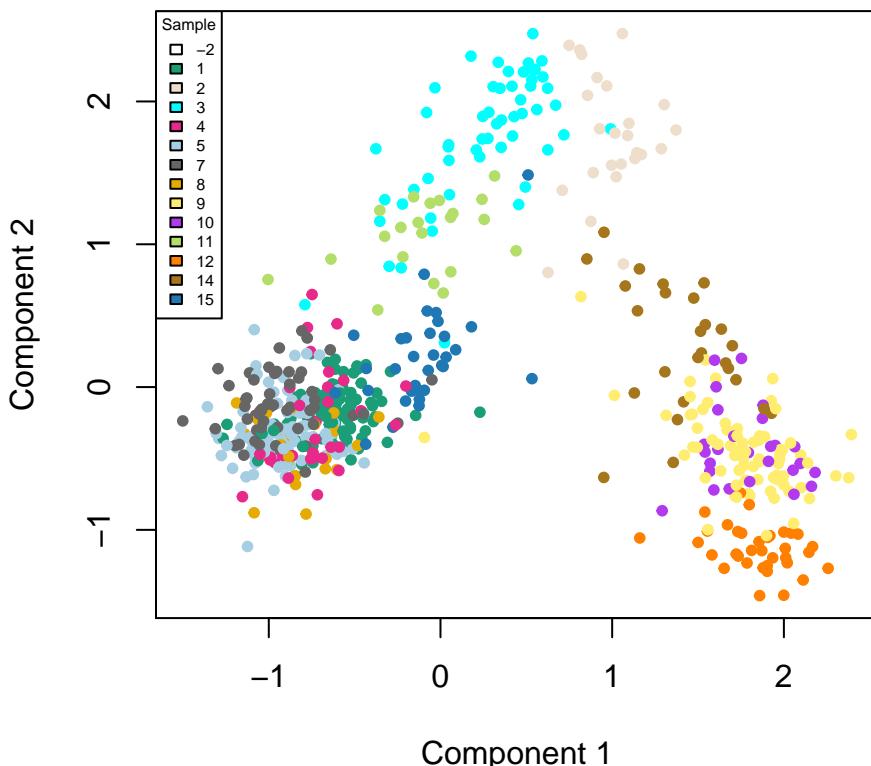


Figure 6. ZINB-WaVE: MDS of the low-dimensional matrix W , where each point represents a cell and cells are color-coded by original published clustering.

Dimensionality reduction

Perhaps the most important component of the results of `zinbwave` is the low-dimensional matrix W , which is stored as the `zinbwave` component of the `reducedDims` slot of the `SingleCellExperiment` object. This matrix can be retrieved using the `reducedDim()` extractor.

```
W <- reducedDim(dataObj)
W[1:3, 1:3]
```

```
##          W1          W2          W3
## OEP01_N706_S501 0.3397236 -0.00975782  0.80744690
## OEP01_N701_S501 0.3066246  0.11110387  0.20854198
## OEP01_N707_S507 0.5364218 -0.05823383 -0.08582661
```

In this workflow, the user-supplied dimension K of the low-dimensional space is set to $K = 50$. The resulting low-dimensional matrix W can be visualized in two dimensions by performing multi-dimensional scaling (MDS) using the Euclidian distance. To verify that W indeed captures the biological signal of interest, we display the MDS results in a scatterplot with colors corresponding to the original published clusters (Figure 6).

```
d <- dist(W)
fit <- cmdscale(d, eig = TRUE, k = 2)
plot(fit$points, col = col_clus[as.character(publishedClusters)], main = "",
      pch = 20, xlab = "Component 1", ylab = "Component 2")
legend(x = "topleft", legend = unique(names(col_clus)), cex = .5,
       fill = unique(col_clus), title = "Sample")
```

Overall, it seems that `zinbwave` was effective at removing batch effects without removing biological signal, in spite of the partial nesting of batches within clusters.

Choice of K

The most important parameter to choose for the dimensionality reduction step is the number of hidden covariates to be included in the model, i.e., the number of columns of the hidden matrix W , which we indicate

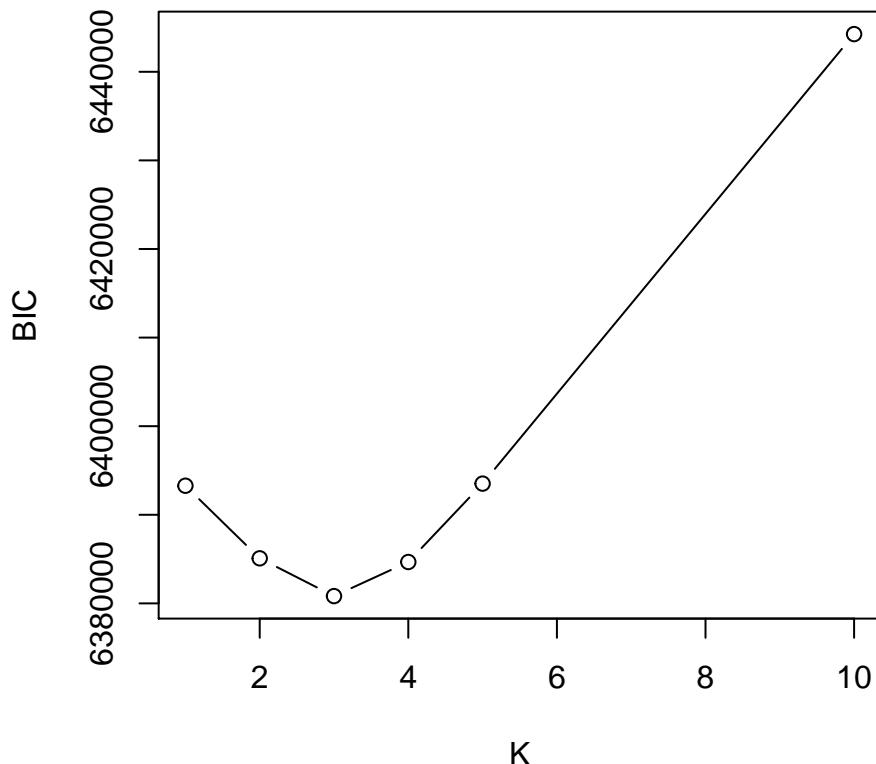
with K. One way to select the optimal value of K is to compare different values in terms of Akaike information criterion (AIC) and Bayesian information criterion (BIC). The `zinbwave` package provides two functions to compute such quantities, `zinbAIC` and `zinbBIC`.

Here, we fit a few different models, specifying a different value of K, using the `zinbFit` function, which, unlike `zinbwave`, returns the estimates of all the parameters of the model, as well as the value of the likelihood function.

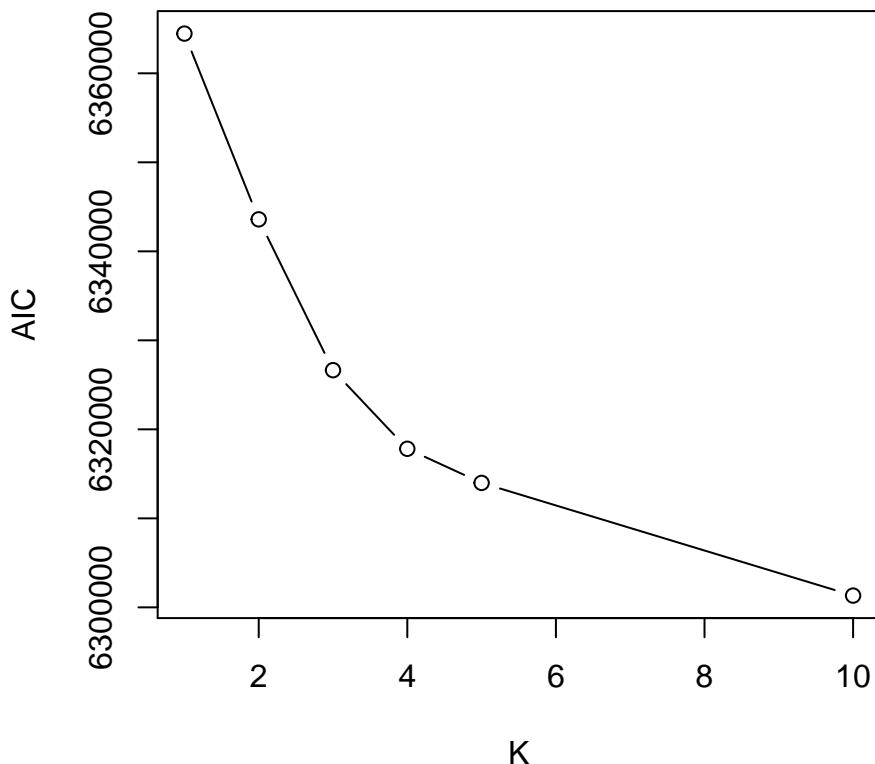
```
k_values <- c(1: 5, 10)

fit <- lapply(k_values, function(k) zinbFit(dataObj, K = k, which_assay = "counts"))
saveRDS(fit, file = "Fit_allK.rds")

k_values <- c(1: 5, 10)
fit <- readRDS("fit_allK.rds")
plot(k_values, sapply(fit, zinbBIC, t(counts(dataObj))),
     ylab = "BIC", xlab = "K", type = "b")
```



```
plot(k_values, sapply(fit, zinbAIC, t(counts(dataObj))),
     ylab = "AIC", xlab = "K", type = "b")
```



In this case, the BIC suggests a value of $K = 3$, while the AIC suggests a value of $K = 10$.

Note, that since ZINB-WaVE is linear in the number of genes and number of cells, and quadratic on K , fitting multiple models and comparing their goodness-of-fit is infeasible for large datasets. In such cases, choosing a single value of K may be needed. In practice, we found that $K=10$ and $K=20$ are reasonable choices in real datasets.

DR: There is quite a difference between AIC and BIC in this case, and it is actually hard to argue one value over another... how should we discussed these results?

Cell clustering: RSEC

The next step of the workflow is to cluster the cells according to the low-dimensional matrix W computed in the previous step. We use the resampling-based sequential ensemble clustering (RSEC) framework implemented in the `RSEC` function from the Bioconductor R package `clusterExperiment`. Specifically, given a set of user-supplied base clustering algorithms and associated tuning parameters (e.g., k -means, with a range of values for k), RSEC generates a collection of candidate clusterings, with the option of resampling cells and using a sequential tight clustering procedure as in [18]. A consensus clustering is obtained based on the levels of co-clustering of samples across the candidate clusterings. The consensus clustering is further condensed by merging similar clusters, which is done by creating a hierarchy of clusters, working up the tree, and testing for differential expression between sister nodes, with nodes of insufficient DE collapsed. As in supervised learning, resampling greatly improves the stability of clusters and considering an ensemble of methods and tuning parameters allows us to capitalize on the different strengths of the base algorithms and avoid the subjective selection of tuning parameters. For more details on the RSEC method and on the `clusterExperiment` package, see [9].

Here, we are applying RSEC to the low-dimensional W matrix from ZINB-WaVE. To do so, we need to specify the `reducedDim` slot to use with the `reduceMethod` option. Here, we skip the merging step to collapse similar clusters based on the amount of differential gene expression between them. In larger dataset, this step may prevent results characterized by a large number of very small clusters (see [9] for details).

```
print(system.time(dataObj <- RSEC(dataObj, k0s = 4:15, alphas = c(0.1),
                                     betas = 0.8, reduceMethod="zinbwave",
                                     clusterFunction = "hierarchical01", minSizes=1,
                                     ncores = NCORES, isCount=FALSE,
                                     dendroReduce="zinbwave",
                                     subsampleArgs = list(resamp.num=100,
                                                          clusterFunction="kmeans",
                                                          clusterArgs=list(nstart=10)),
```

```
verbose=TRUE,
consensusProportion = 0.7,
mergeMethod = "none", random.seed=424242,
consensusMinSize = 10)))
```

```
##      user    system   elapsed
## 1023.594   99.627  215.421
```

```
save(dataObj, file = 'dataObj_afterRSEC_k10.rda')
#load('dataObj_afterRSEC.rda')
```

The correspondance between the clusters we found here and the original clusters is as follows.

```
table(data.frame(original = publishedClusters, ours = primaryClusterNamed(dataObj)))

##          ours
## original -1 c1 c10 c11 c12 c13 c14 c15 c16 c17 c18 c19 c2 c20 c21 c22 c23
##       -2 53  4   3  10  22   7  14   3   0   6   0  10   2   1   0   0   0   0   0
##        1 18  0  16   0   0   0   1   5   6  10  10   0   0   0   0   0  11   0  10
##        2  1  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##        3  3  0   0   1   0   0   0   0   0   0   0   0   0   0   1   0   0   0   5   0
##        4  3  9  0   0   0   0   0   1   0   0   0   0   0   0   2   0   4   0   0   0
##        5 28  3   8   0   1   0   0   4  10   1   4   0   0   1   1   0   0   1   1   0
##        7  7 35  0   0   0   0   3   0   0   0   0   0   2   0   7   0   0   0   0   0
##        8  9  0   0   0   0   0   0   7   2   0   3   0   0   0   0   0   0   0   0   0
##        9  4  0   0   0   0 12   0   0   0   0   0   1   48   0   0   0   0   0   0   0
##       10  1  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##       11  0  0   0 13   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   6   0
##       12  0  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##       14  4  0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##       15  2  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0

##          ours
## original c3 c4 c5 c6 c7 c8 c9
##       -2  2  0  0  1  2  1 10
##        1  0  0  0  3  0  0  0
##        2  0  0  0  0  0 24  0
##        3  0  0 40  0  0  4  0
##        4  0  0  0  0  0  0 16
##        5  0  0  0 32  0  0  0
##        7  0  0  0  0  0  0  4
##        8  0  0  0  6  0  0  0
##        9  1  8  0  0  0  0  0
##       10 25  0  0  0  0  0  0
##       11  0  0  2  0  0  0  0
##       12  0 35  0  0  0  0  0
##       14 20  0  0  0  0  1  0
##       15  0  0  0  0 29  1  0
```

To facilitate the comparison between our results and the original clusters, we set a color scheme that allows a direct visual comparison with Figure 2. We can change the clustering color scheme using `clusterExperiment's` `recolorClusters` function.

```
new_col <- col_clus[c("1", "4", "12", "3", "10", "15", "9")]
names(new_col) <- paste0("c", 1:7)
dataObj <- recolorClusters(dataObj, value = new_col)
```

The resulting candidate clusterings can be visualized using the `plotClusters` function (Figure 7), where columns correspond to cells and rows to different clusterings. Each sample is color-coded based on its clustering for that row, where the colors have been chosen to try to match up clusters that show large overlap across rows. The first row correspond to a consensus clustering across all candidate clusterings.

Cluster name	Description	Color	Correspondence
c1	HBC	green	original 1, 5
c2	mSUS	magenta	original 4, 7
c3	mOSN	orange	original 9, 12
c4	GBC	cyan	original 2, 3, 11
c5	Immature Neuron	purple	original 10, 14
c6	MV	blue	original 15
c7	mOSN	yellow	original 9

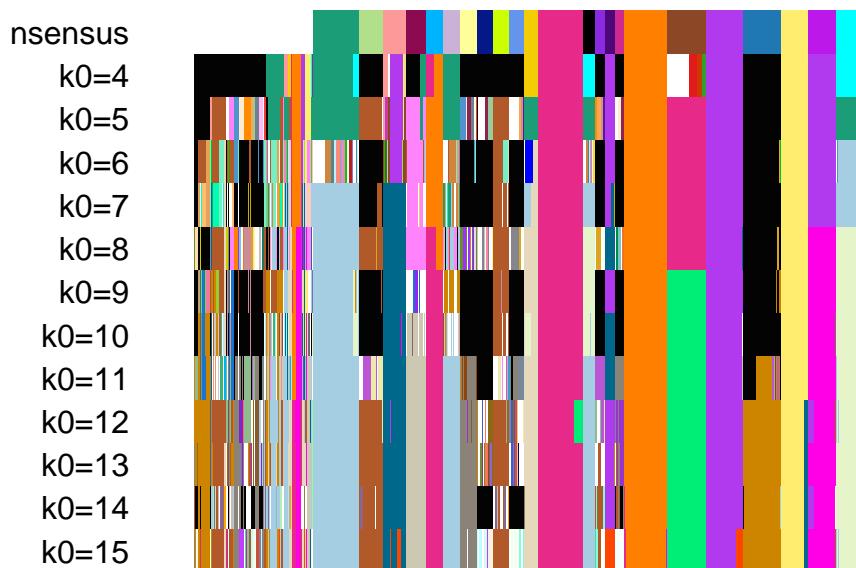


Figure 7. RSEC: Candidate clusterings found using the function RSEC from the clusterExperiment package. The top row corresponds to the consensus clustering and is color-coded accordingly. The other rows correspond to the individual clustering across which the consensus was computed. In each row, the color is chosen to match as close as possible the color of the cluster from the previous row with the largest overlap.

```
plotClusters(dataObj, existingColors = "firstOnly")
```

The `plotCoClustering` function produces a heatmap of the co-clustering matrix, which records, for each pair of cells, the proportion of times they were clustered together across the candidate clusters (Figure 8).

```
plotCoClustering(dataObj)
```

The distribution of cells across the consensus clusters can be visualized in Figure 9 and is as follows:

```
table(primaryClusterNamed(dataObj))
```

```
## 
## -1 c1 c10 c11 c12 c13 c14 c15 c16 c17 c18 c19 c2 c20 c21 c22 c23 c3
## 133 51 27 25 23 19 19 19 18 17 17 16 50 13 12 11 10 48
## c4 c5 c6 c7 c8 c9
## 43 42 42 31 31 30
```

```
plotBarplot(dataObj, legend = FALSE)
```

The distribution of cells in our workflow's clustering overall agrees with that in the original published clustering (Figure 10 and 11), the main difference being that several of the published clusters were merged here into single clusters. This discrepancy is likely caused by the fact that we started with the top 1,000 genes, which might not be enough to discriminate between closely related clusters.

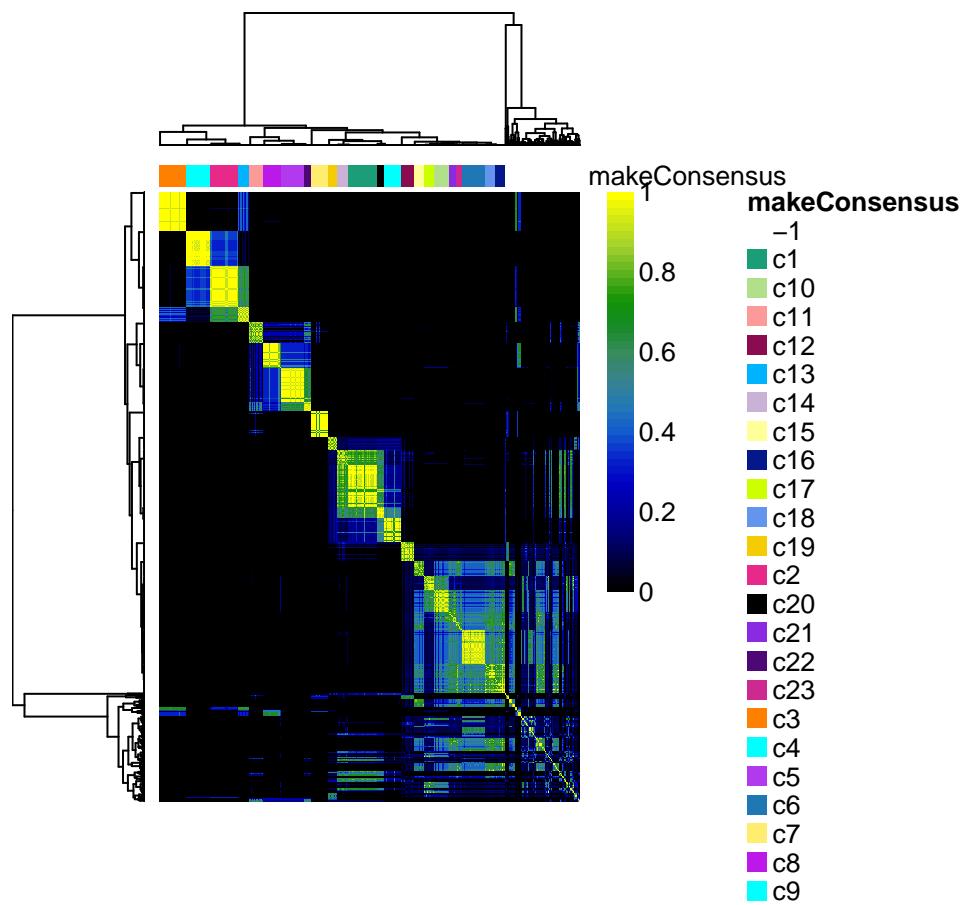


Figure 8. RSEC: Heatmap of co-clustering matrix.

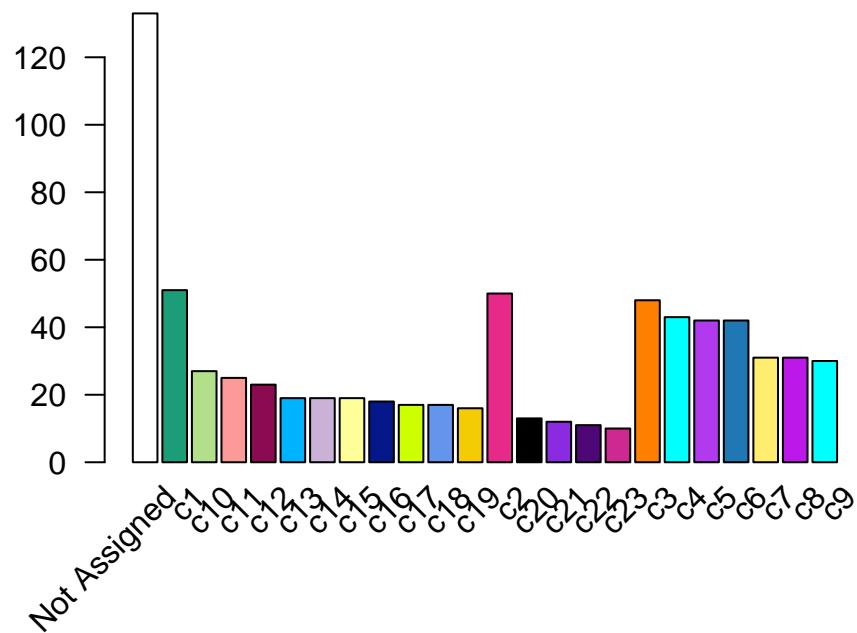


Figure 9. RSEC: Barplot of number of cells per cluster for our workflow's RSEC clustering.

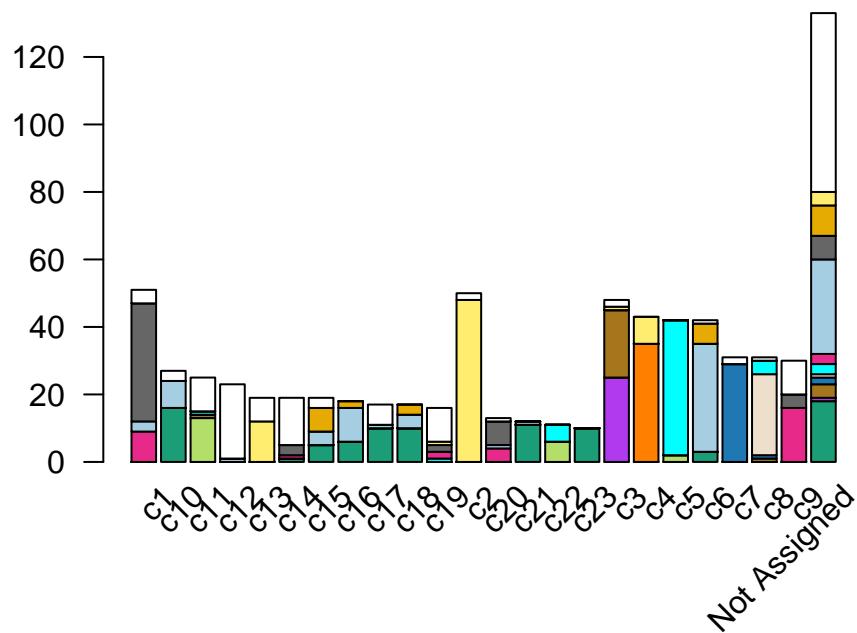


Figure 10. RSEC: Barplot of number of cells per cluster, for our workflow's RSEC clustering, color-coded by original published clustering.

```

dataObj <- addClusterings(dataObj, colData(dataObj)$publishedClusters,
                           clusterLabel = "publishedClusters")

## change default color to match with Figure 7
clusterLegend(dataObj)$publishedClusters[, "color"] <-
  col_clus[clusterLegend(dataObj)$publishedClusters[, "name"]]

plotBarplot(dataObj, whichClusters=c("makeConsensus","publishedClusters"),
            xlab = "", legend = FALSE,missingColor="white")

```

```

plotClustersTable(dataObj, whichClusters=c("makeConsensus","publishedClusters"))

```

Figure 12 displays a heatmap of the normalized expression measures for the 1,000 most variable genes, where cells are clustered according to the RSEC consensus.

```

# Set colors for additional sample data
experimentColors <- bigPalette[1:nlevels(colData(dataObj)$Experiment)]
batchColors <- bigPalette[1:nlevels(colData(dataObj)$Batch)]
metaColors <- list("Experiment" = experimentColors,
                   "Batch" = batchColors)

plotHeatmap(dataObj,
            whichClusters = c("makeConsensus","publishedClusters"),
            clusterFeaturesData = "all",
            clusterSamplesData = "dendrogramValue", breaks = 0.99,
            colData = c("Batch", "Experiment"),
            clusterLegend = metaColors, annLegend = FALSE, main = "")

```

Finally, we can visualize the cells in a two-dimensional space using the MDS of the low-dimensional matrix W and coloring the cells according to their newly-found RSEC clusters (Figure 13); this is analogous to Figure 6 for the original published clusters.

```

plotReducedDims(dataObj, whichCluster="primary", reducedDim="zinbwave", pch=20,
                 xlab = "Component1", ylab = "Component2", legendTitle="Sample", main="",
                 plotUnassigned=FALSE
)

```

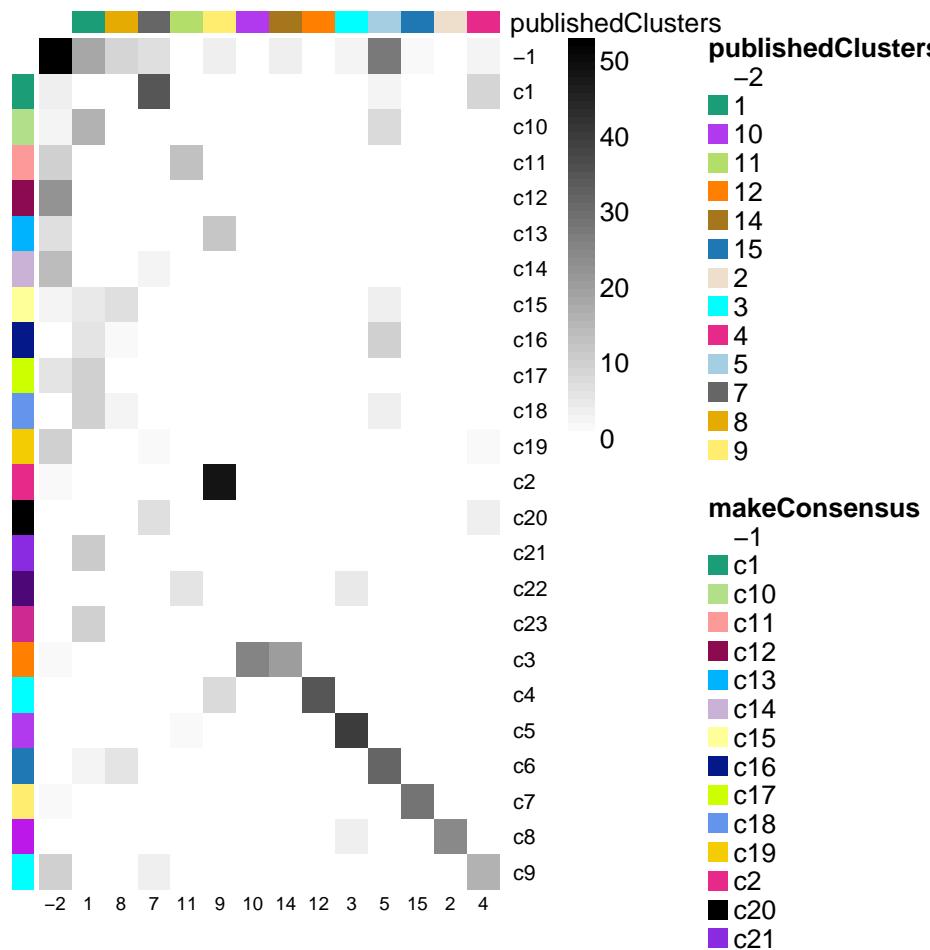


Figure 11. RSEC: Confusion matrix of number of cells per cluster, for our workflow's RSEC clustering and the original published clustering.

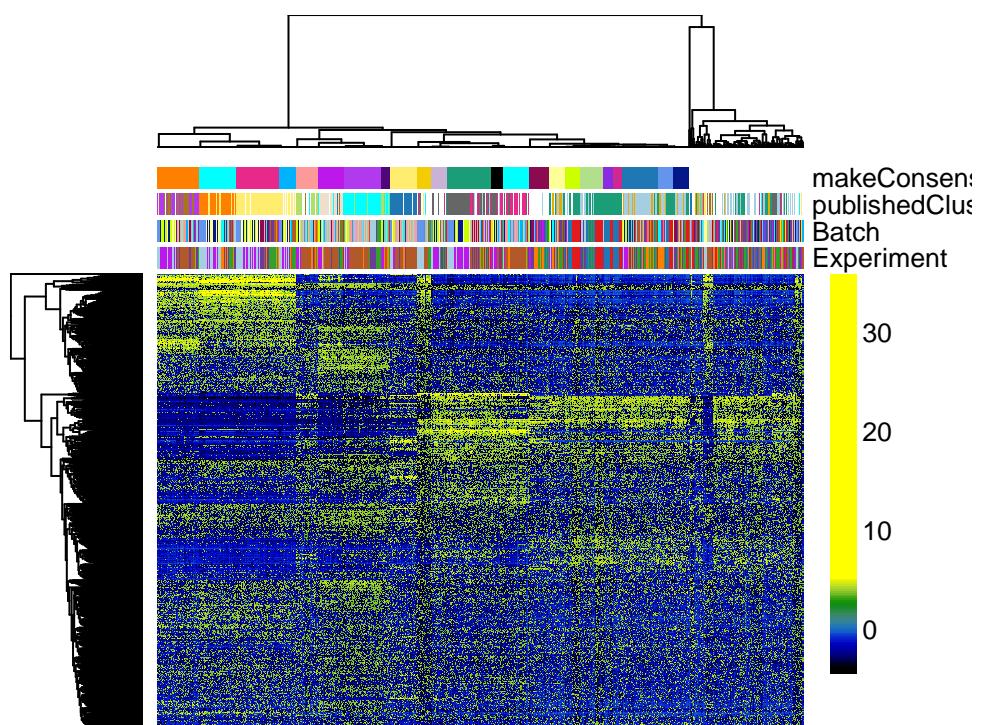


Figure 12. RSEC: Heatmap of the normalized expression measures for the 1,000 most variable genes, where rows correspond to genes and columns to cells ordered by RSEC clusters.

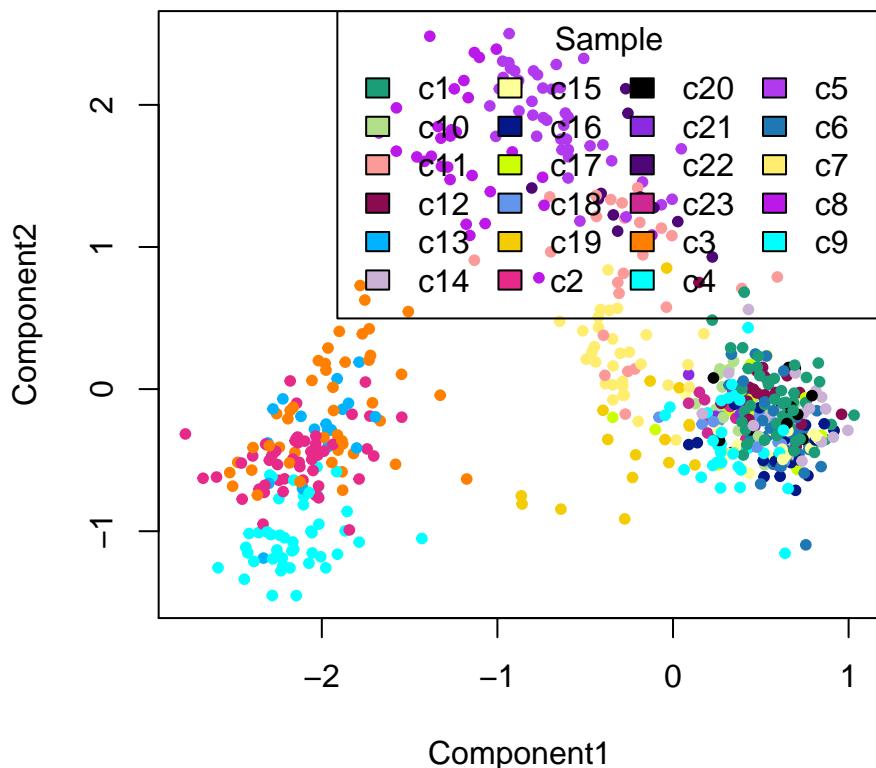


Figure 13. RSEC: MDS of the low-dimensional matrix W , where each point represents a cell and cells are color-coded by RSEC clustering.

Cell lineage and pseudotime inference: Slingshot

We now demonstrate how to use the Bioconductor package `slingshot` to infer branching cell lineages and order cells by developmental progression along each lineage. The method, proposed in [10], comprises two main steps: (1) The inference of the global lineage structure (i.e., the number of lineages and where they branch) using a minimum spanning tree (MST) on the clusters identified above by RSEC and (2) the inference of cell pseudotime variables along each lineage using a novel method of simultaneous principal curves. The approach in (1) allows the identification of any number of novel lineages, while also accommodating the use of domain-specific knowledge to supervise parts of the tree (e.g., known terminal states); the approach in (2) yields robust pseudotimes for smooth, branching lineages.

This analysis is performed by the `slingshot` function and the results are stored in a `SlingshotDataSet` object. The minimal input to this function is a low-dimensional representation of the cells and a set of cluster labels; these can be separate objects (ie. a matrix and a vector) or, as below, components of a `SingleCellExperiment` object. When a `SingleCellExperiment` object is provided as input, the output will be an updated `SingleCellExperiment` object containing a `SlingshotDataSet` as an element of the `int_metadata` list, which can be accessed through the `SlingshotDataSet` function. For more low-level control of the lineage inference procedure, the two steps of fitting the MST and the simultaneous principal curves may be run separately via the functions `getLineages` and `getCurves`.

From the original published work, we know that the start cluster should correspond to HBCs and the end clusters to MV, mOSN, and mSUS cells. Additionally, we know that GBCs should be at a junction before the differentiation between MV and mOSN cells (Figure 2).

To infer lineages and pseudotimes, we apply Slingshot to the 4-dimensional MDS of the low-dimensional matrix W . We found that the Slingshot results were robust to the number of dimensions k for the MDS (we tried k from 2 to 5). Here, we use the unsupervised version of Slingshot, where we only provide the identity of the start cluster but not of the end clusters.

```
pseudoCe <- dataObj[, !primaryClusterNamed(dataObj) %in% c("-1")]
X <- reducedDim(pseudoCe, type = "zinbwave")
mds <- cmdscale(dist(X), eig = TRUE, k = 4)
lineages <- slingshot(mds$points, clusterLabels = primaryClusterNamed(pseudoCe), start.clus = "c1")
```

Before discussing the simultaneous principal curves, we examine the global structure of the lineages by plotting the MST on the clusters. This shows that our implementation has recovered the lineages found in the published

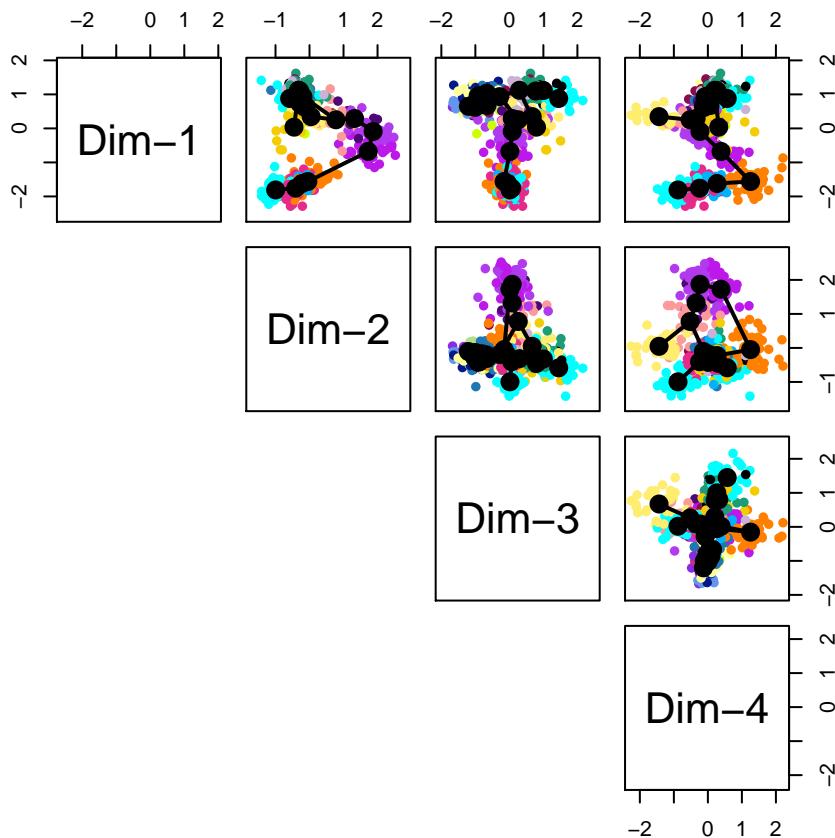


Figure 14. Slingshot: Cells color-coded by cluster in a 4-dimensional MDS space, with connecting lines between cluster centers representing the inferred global lineage structure.

work (Figure 14). The `slingshot` package also includes functionality for 3-dimensional visualization as in Figure 2, using the `plot3d` function from the package `rgl`.

```
colorCl<-convertClusterLegend(pseudoCe,whichCluster="primary",output="matrixColors")[,1]
pairs(lineages, type="lineages", col = colorCl)
```

Having found the global lineage structure, `slingshot` then constructed a set of smooth, branching curves in order to infer the corresponding pseudotime variables. Simultaneous principal curves are constructed from the individual cells along each lineage, rather than the cell clusters. During this iterative process, a cell may even be reassigned to a different lineage if it is significantly closer to the corresponding curve. This makes `slingshot` less reliant on the original clustering and generally more stable. The final curves are shown in Figure 15.

```
pairs(lineages, type="curves", col = colorCl)
```

```
lineages
```

```
## class: SlingshotDataSet
##
## Samples Dimensions
##       614          4
##
## lineages: 7
## Lineage1: c1  c14  c12  c11  c22  c5   c8   c3   c13  c2   c4
## Lineage2: c1  c14  c12  c17  c10  c18  c15  c16
## Lineage3: c1  c14  c12  c17  c10  c18  c15  c6
## Lineage4: c1  c14  c12  c17  c10  c23  c21
## Lineage5: c1  c14  c12  c11  c7
## Lineage6: c1  c14  c19
## Lineage7: c1  c20  c9
```

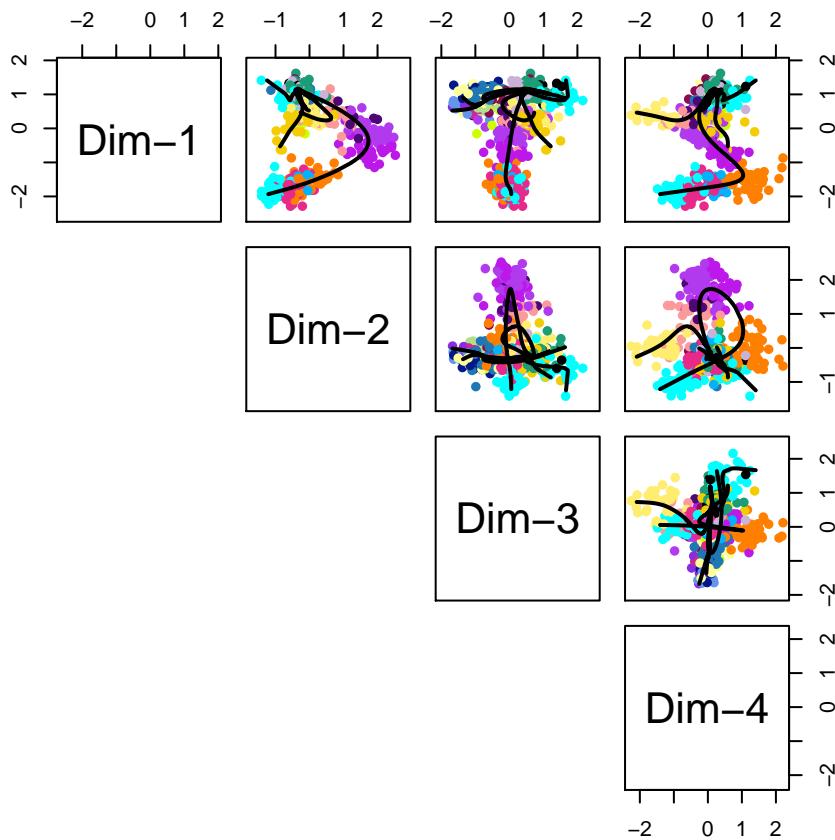


Figure 15. Slingshot: Cells color-coded by cluster in a 4-dimensional MDS space, with smooth curves representing each inferred lineage.

```
##  
## curves: 7  
## Curve1: Length: 8.5132 Samples: 329.87  
## Curve2: Length: 2.7948 Samples: 222.85  
## Curve3: Length: 4.744 Samples: 218.78  
## Curve4: Length: 3.1019 Samples: 202.44  
## Curve5: Length: 4.4678 Samples: 137.99  
## Curve6: Length: 3.1305 Samples: 89.27  
## Curve7: Length: 3.4457 Samples: 101.52
```

As an alternative, we could have incorporated the MDS results into the `pseudoCe` object and applied `slingshot` directly to it. Here, we need to specify that we want to use the MDS results, because `slingshot` would otherwise use the first element of the `reducedDims` list (in this case, the 10-dimensional `W` matrix from `zinbwave`).

```
reducedDim(pseudoCe, "MDS") <- mds$points  
pseudoCe <- slingshot(pseudoCe, reducedDim = "MDS", start.clus = "c1")  
pseudoCe
```

```
## class: ClusterExperiment  
## dim: 1000 614  
## reducedDimNames: zinbwave MDS  
## filterStats: var  
## -----  
## Primary cluster type: makeConsensus  
## Primary cluster label: makeConsensus  
## Table of clusters (of primary clustering):  
##   c1 c10 c11 c12 c13 c14 c15 c16 c17 c18 c19   c2 c20 c21 c22 c23   c3   c4  
##   51  27  25  23  19  19  19  18  17  17  16  50  13  12  11  10  48  43  
##   c5   c6   c7   c8   c9  
##   42   42   31   31   30  
## Total number of clusterings: 14  
## No dendrogram present
```

```

## -----
## Workflow progress:
## clusterMany run? Yes
## makeConsensus run? Yes
## makeDendrogram run? No
## mergeClusters run? No

colData(pseudoCe) [,grep("^sling", colnames(colData(pseudoCe)))]
```

```

## DataFrame with 614 rows and 8 columns
##               slingClusters slingPseudotime_1 slingPseudotime_2
##               <character>      <numeric>      <numeric>
## OEP01_N706_S501       c21          NA          NA
## OEP01_N705_S501       c17          NA  1.93127352404575
## OEP01_N702_S505       c15          NA  2.50670374710029
## OEP01_N703_S501       c23          NA  2.29798579428116
## OEP01_N708_S503       c21          NA          NA
## ...
## ...
## OEL23_N704_S510       c19          NA          NA
## OEL23_N705_S502       c9           NA          NA
## OEL23_N706_S502       c4   7.40996381434697
## OEL23_N704_S503       c4   7.78759629295255
## OEL23_N703_S502       c1  0.502033570743035  0.512602438841934
##               slingPseudotime_3 slingPseudotime_4 slingPseudotime_5
##               <numeric>      <numeric>      <numeric>
## OEP01_N706_S501        NA  3.10186714922018          NA
## OEP01_N705_S501    1.82492837534338  1.9440410411311  1.62527999787044
## OEP01_N702_S505        NA  2.79972175456208          NA
## OEP01_N703_S501    2.05200396213798  2.3459550504826          NA
## OEP01_N708_S503        NA  2.45562639391915          NA
## ...
## ...
## OEL23_N704_S510        NA          NA          NA
## OEL23_N705_S502        NA          NA          NA
## OEL23_N706_S502        NA          NA          NA
## OEL23_N704_S503        NA          NA          NA
## OEL23_N703_S502    0.5157425855015  0.516165562034881  0.509585179283938
##               slingPseudotime_6 slingPseudotime_7
##               <numeric>      <numeric>
## OEP01_N706_S501        NA          NA
## OEP01_N705_S501        NA          NA
## OEP01_N702_S505        NA          NA
## OEP01_N703_S501        NA          NA
## OEP01_N708_S503        NA          NA
## ...
## ...
## OEL23_N704_S510    2.09221605566682          NA
## OEL23_N705_S502        NA  1.33175327120853
## OEL23_N706_S502        NA          NA
## OEL23_N704_S503        NA          NA
## OEL23_N703_S502    0.543064962770762  0.544042565927638
```

The result of `slingshot` applied to a `ClusterExperiment` object is still of class `ClusterExperiment`. Note that we did not specify a set of cluster labels, implying that `slingshot` should use the default `primaryClusterNamed` vector.

In the workflow, we recover the right ordering of the clusters using the unsupervised version of `slingshot`. However, in some other cases, we have noticed that we need to give more guidance to the algorithm to find the correct ordering. `slingshot` has the option for the user to provide known end cluster(s). Here is the code to use `slingshot` in a supervised setting, where we know that clusters `c3` and `c7` represent terminal cell fates.

```
pseudoCeSup <- slingshot(pseudoCe, reducedDim = "MDS", start.clus = "c1",
                           end.clus = c("c3", "c6", "c2"))
```

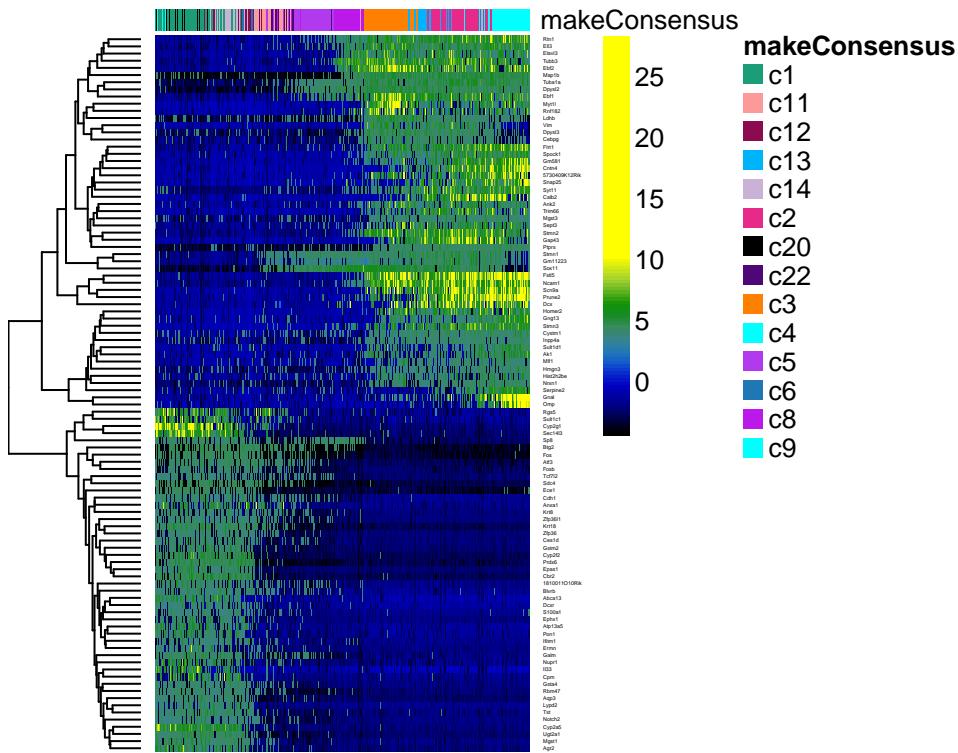


Figure 16. DE: Heatmap of the normalized expression measures for the 100 most significantly DE genes for the neuronal lineage, where rows correspond to genes and columns to cells ordered by pseudotime.

Differential expression analysis along lineages

After assigning the cells to lineages and ordering them within lineages, we are interested in finding genes that have non-constant expression patterns over pseudotime.

More formally, for each lineage, we use the robust local regression method loess to model in a flexible, non-linear manner the relationship between a gene's normalized expression measures and pseudotime. We then can test the null hypothesis of no change over time for each gene using the gam package.

Generalized Additive Models (GAMs) [19] are a class of models that replaces the usual linear function of the coefficients with a sum of smooth functions. In the code below, we use a special case of the GAM in which only one covariate (pseudotime) is present. This correspond to a simple loess regression, but with the added advantage of providing a framework for the testing of the significance of the association between pseudotime and expression.

We implement this approach for the neuronal lineage and display the expression measures of the top 100 genes by p-value in the heatmap of Figure 16.

```
t <- colData(pseudoCe)$slingPseudotime_1
y <- transformData(pseudoCe)
gam.pval <- apply(y, 1, function(z){
  d <- data.frame(z=z, t=t)
  tmp <- gam(z ~ lo(t), data=d)
  p <- summary(tmp)[4][[1]][1,5]
  p
})

topgenes <- names(sort(gam.pval, decreasing = FALSE))[1:100]

pseudoCe1 <- pseudoCe[, !is.na(t)]
orderSamples(pseudoCe1) <- order(t[!is.na(t)])

plotHeatmap(pseudoCe1[topgenes,], clusterSamplesData = "orderSamplesValue",
            breaks = .99)
```

DR: here, describe the transformData and plotHeatmap function. Above, probably right after RSEC, explain the ClusterExperiment class and its relation to SCE.

Differential expression between clusters

DR: I think that it will be nice to showcase the `getBestFeatures` function here as well as our observational weights.

Further developments

DR: Not sure if we need this section. The previous future developments are now done (nice!). Perhaps here we should talk about scalability and HDF5? Kelly, is there anything you want to add here on DE along lineages?

Conclusions

This workflow provides a tutorial for the analysis of scRNA-seq data in R/Bioconductor. It covers four main steps: (1) dimensionality reduction accounting for zero inflation and over-dispersion and adjusting for gene and cell-level covariates; (2) robust and stable cell clustering using resampling-based sequential ensemble clustering; (3) inference of cell lineages and ordering of the cells by developmental progression along lineages; and (4) DE analysis along lineages. The workflow is general and flexible, allowing the user to substitute the statistical method used in each step by a different method. We hope our proposed workflow will ease technical aspects of scRNA-seq data analysis and help with the discovery of novel biological insights.

Alternative approaches

Many different methods have been proposed for the analysis of scRNA-seq data and this paper presents only one possible workflow, using tools that we have developed in recent years. The use of the `SingleCellExperiment` class, which is the shared “language” among Bioconductor’s scRNA-seq software packages, makes it extremely easy to change one of the step above to use a different approach. For instance, one could use `scPipe` [11] for gene expression quantification, `scater` [2] for quality control and filtering, `scran` [20] to robustly calculate global scaling factors for normalization, and `SC3` [21] for clustering, with minimal to the code structure above. This makes the present workflow a truly modular approach, in which a user can use their favorite tool for each step with the guarantee that the output of the step will be readable by the next step.

An alternative normalization strategy is to compute gene and cell-specific scaling factors (e.g., `SCNorm` [22]). Similarly, `monocle` [23] or `TSCAN` [24] for cell lineage inference. Finally, the software suites `Seurat` [25] (available on CRAN) and `scde` [26] provide alternative pipelines for cell type discovery, differential expression, and gene set analysis. A comprehensive list of tools developed for the analysis of scRNA-seq data is available through at <https://www.scrna-tools.org> [27].

Software availability

This section will be generated by the Editorial Office before publication. Authors are asked to provide some initial information to assist the Editorial Office, as detailed below.

1. URL link to where the software can be downloaded from or used by a non-coder (AUTHOR TO PROVIDE; optional)
2. URL link to the author’s version control system repository containing the source code (AUTHOR TO PROVIDE; required)
3. Link to source code as at time of publication (*F1000Research* TO GENERATE)
4. Link to archived source code as at time of publication (*F1000Research* TO GENERATE)
5. Software license (AUTHOR TO PROVIDE; required)

The source code for this package can be found at <https://github.com/fperraudeau/singlecellworkflow>. The four packages used in the workflow (`scone`, `zinbwave`, `clusterExperiment`, and `slingshot`) are Bioconductor R packages and are available at, respectively, <https://bioconductor.org/packages/scone>, <https://bioconductor.org/packages/zinbwave>, <https://bioconductor.org/packages/clusterExperiment>, and <https://bioconductor.org/packages/slingshot>.

```
sessionInfo()
```

```

## R version 3.5.0 (2018-04-23)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS High Sierra 10.13.6
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] splines    parallel   stats4     stats      graphics   grDevices utils
## [8] datasets   methods    base
##
## other attached packages:
## [1] RColorBrewer_1.1-2          gam_1.15
## [3] doParallel_1.0.11           iterators_1.0.10
## [5] foreach_1.4.4               slingshot_0.99.12
## [7] prcurve_2.1.0               zinbwave_1.3.3-9001
## [9] scone_1.5.0                clusterExperiment_2.1.5
## [11] bigmemory_4.5.33            SingleCellExperiment_1.3.9
## [13] SummarizedExperiment_1.11.6 DelayedArray_0.7.19
## [15] matrixStats_0.53.1          Biobase_2.41.2
## [17] GenomicRanges_1.33.9        GenomeInfoDb_1.17.1
## [19] IRanges_2.15.16             S4Vectors_0.19.19
## [21] BiocGenerics_0.27.1         BiocParallel_1.15.8
## [23] knitr_1.20                 BiocStyle_2.9.3
##
## loaded via a namespace (and not attached):
## [1] R.utils_2.6.0                  tidyselect_0.2.4
## [3] htmlwidgets_1.2                RSQLite_2.1.1
## [5] AnnotationDbi_1.43.1          grid_3.5.0
## [7] trimcluster_0.1-2.1            RNeXML_2.1.1
## [9] devtools_1.13.6               DESeq_1.33.0
## [11] munsell_0.5.0                 codetools_0.2-15
## [13] miniUI_0.1.1.1               withr_2.1.2
## [15] colorspace_1.3-2              energy_1.7-4
## [17] uuid_0.1-2                   rstudioapi_0.7
## [19] pspline_1.0-18               robustbase_0.93-1
## [21] bayesm_3.1-0.1               NMF_0.21.0
## [23] git2r_0.23.0                GenomeInfoDbData_1.1.0
## [25] hwriter_1.3.2                bit64_0.9-7
## [27] rhdf5_2.25.4                rprojroot_1.3-2
## [29] xfun_0.3                     EDASeq_2.15.3
## [31] diptest_0.75-7               R6_2.2.2
## [33] locfit_1.5-9.1               manipulateWidget_0.10.0
## [35] flexmix_2.3-14              bitops_1.0-6
## [37] assertthat_0.2.0              promises_1.0.1
## [39] scales_0.5.0                 nnet_7.3-12
## [41] gtable_0.2.0                 phylobase_0.8.4
## [43] RUVSeq_1.15.1                rlang_0.2.1
## [45] genefilter_1.63.0             rtracklayer_1.41.3
## [47] lazyeval_0.2.1               hexbin_1.27.2
## [49] rgl_0.99.16                  yaml_2.1.19
## [51] reshape2_1.4.3                crosstalk_1.0.0
## [53] GenomicFeatures_1.33.0       backports_1.1.2
## [55] httpuv_1.4.4.2               tensorA_0.36
## [57] tools_3.5.0                  bookdown_0.7
## [59] gridBase_0.4-7                ggplot2_3.0.0
## [61] gplots_3.0.1                 stabledist_0.7-1
## [63] Rcpp_0.12.17                 plyr_1.8.4
## [65] progress_1.2.0                zlibbioc_1.27.0
## [67] purrrr_0.2.5                 RCurl_1.95-4.11
## [69] prettyunits_1.0.2             viridis_0.5.1

```

```

## [71] cluster_2.0.7-1           magrittr_1.5
## [73] data.table_1.11.4         RSpectra_0.13-1
## [75] mvtnorm_1.0-8            whisker_0.3-2
## [77] gsl_1.9-10.3             aroma.light_3.11.0
## [79] mime_0.5                 hms_0.4.2
## [81] evaluate_0.11             xtable_1.8-2
## [83] XML_3.98-1.12            mclust_5.4.1
## [85] gridExtra_2.3              compiler_3.5.0
## [87] biomaRt_2.37.3            tibble_1.4.2
## [89] KernSmooth_2.23-15        crayon_1.3.4
## [91] R.oo_1.22.0                htmltools_0.3.6
## [93] later_0.7.3                segmented_0.5-3.0
## [95] pcaPP_1.9-73               BiocWorkflowTools_1.7.2
## [97] tidyR_0.8.1                geneplotter_1.59.0
## [99] howmany_0.3-1              DBI_1.0.0
## [101] MASS_7.3-50               fpc_2.1-11.1
## [103] boot_1.3-20               compositions_1.40-2
## [105] ShortRead_1.39.0          Matrix_1.2-14
## [107] ade4_1.7-11               R.methodsS3_1.7.1
## [109] gdata_2.18.0              igraph_1.2.1
## [111] bindr_0.1.1                pkgconfig_2.0.1
## [113] bigmemory.sri_0.1.3        rncl_0.8.2
## [115] GenomicAlignments_1.17.3   registry_0.5
## [117] numDeriv_2016.8-1          locfdr_1.1-8
## [119] xml2_1.2.0                rARPACK_0.11-0
## [121] annotate_1.59.1            rngtools_1.3.1
## [123] webshot_0.5.0              pkgmaker_0.27
## [125] XVector_0.21.3             bibtex_0.4.2
## [127] stringr_1.3.1              digest_0.6.15
## [129] copula_0.999-18            ADGofTest_0.3
## [131] softImpute_1.4              Biostrings_2.49.0
## [133] rmarkdown_1.10               dendextend_1.8.0
## [135] edgeR_3.23.3               kernlab_0.9-26
## [137] shiny_1.1.0                 Rsamtools_1.33.2
## [139] gtools_3.8.1                modeltools_0.2-22
## [141] jsonlite_1.5                nlme_3.1-137
## [143] bindrcpp_0.2.2              Rhdf5lib_1.3.1
## [145] viridisLite_0.3.0            limma_3.37.3
## [147] pillar_1.3.0                lattice_0.20-35
## [149] httr_1.3.1                 DEoptimR_1.0-8
## [151] survival_2.42-6              glue_1.3.0
## [153] prabclus_2.2-6              glmnet_2.0-16
## [155] bit_1.1-14                  mixtools_1.1.0
## [157] class_7.3-14                stringi_1.2.3
## [159] HDF5Array_1.9.5              blob_1.1.1
## [161] latticeExtra_0.6-28          caTools_1.17.1
## [163] memoise_1.1.0                dplyr_0.7.6
## [165] ape_5.1

```

Author contributions

FP, DR, KS, and EP performed the data analysis and wrote the code portions of the workflow. DR, FP, and SD wrote the text portion of the workflow, with contributions from the other three authors. DR, EP, and SD supervised the research.

Competing interests

No competing interests were disclosed.

Grant information

DR, KS, EP, and SD were supported by the National Institutes of Health BRAIN Initiative (U01 MH105979, PI: John Ngai). KS was supported by a training grant from the National Human Genome Research Institute (T32000047).

DR: Update grant information!

Acknowledgments

The authors are grateful to Professor John Ngai (Department of Molecular and Cell Biology, UC Berkeley) and his group members Dr. Russell B. Fletcher and Diya Das for motivating the research presented in this workflow and for valuable feedback on applications to biological data. We would also like to thank Michael B. Cole for his contributions to **scone**.

References

- [1] A Lun, D McCarthy, and J Marioni. A step-by-step workflow for low-level analysis of single-cell RNA-seq data with Bioconductor [version 2; referees: 3 approved, 2 approved with reservations]. *F1000Research*, 5(2122), 2016. doi: 10.12688/f1000research.9501.2.
- [2] Davis 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*, page btw777, jan 2017. ISSN 1367-4803. doi: 10.1093/bioinformatics/btw777. URL <https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/btw777>.
- [3] Wolfgang Huber, Vincent J Carey, Robert Gentleman, Simon Anders, Marc Carlson, Benilton S Carvalho, Hector Corrada Bravo, Sean Davis, Laurent Gatto, Thomas Girke, Raphael Gottardo, Florian Hahne, Kasper D Hansen, Rafael A Irizarry, Michael Lawrence, Michael I Love, James MacDonald, Valerie Obenchain, Andrzej K Oleś, Hervé Pagès, Alejandro Reyes, Paul Shannon, Gordon K Smyth, Dan Tenenbaum, Levi Waldron, and Martin Morgan. Orchestrating high-throughput genomic analysis with Bioconductor. *Nature Methods*, 12(2):115–121, jan 2015. ISSN 1548-7091. doi: 10.1038/nmeth.3252. URL <http://www.nature.com/doifinder/10.1038/nmeth.3252>.
- [4] Aaron Lun and Davide Risso. *SingleCellExperiment: S4 Classes for Single Cell Data*, 2018. R package version 1.3.6.
- [5] Wolfgang Huber, Vincent J Carey, Robert Gentleman, Simon Anders, Marc Carlson, Benilton S Carvalho, Hector Corrada Bravo, Sean Davis, Laurent Gatto, Thomas Girke, et al. Orchestrating high-throughput genomic analysis with bioconductor. *Nature Methods*, 12(2):115, 2015.
- [6] Martin Morgan, Valerie Obenchain, Jim Hester, and Hervé Pagès. *SummarizedExperiment: SummarizedExperiment container*, 2018. R package version 1.11.5.
- [7] Russell B Fletcher, Diya Das, Levi Gadye, Kelly N Street, Ariane Baudhuin, Allon Wagner, Michael B Cole, Quetzal Flores, Yoon Gi Choi, Nir Yosef, Elizabeth Purdom, Sandrine Dudoit, Davide Risso, and John Ngai. Deconstructing Olfactory Stem Cell Trajectories at Single-Cell Resolution. *Cell Stem Cell*, 20(6):817–830.e8, jun 2017. ISSN 1934-5909. doi: 10.1016/j.stem.2017.04.003. URL <http://dx.doi.org/10.1016/j.stem.2017.04.003>.
- [8] Davide Risso, Fanny Perraudeau, Svetlana Gribkova, Sandrine Dudoit, and Jean-Philippe Vert. A general and flexible method for signal extraction from single-cell rna-seq data. *Nature Communications*, 9(1):284, 2018.
- [9] Davide Risso, Liam Purvis, Russell Fletcher, Diya Das, John Ngai, Sandrine Dudoit, and Elizabeth Purdom. clusterExperiment and RSEC: A Bioconductor package and framework for clustering of single-cell and other large gene expression datasets. *bioRxiv*, page 280545, 2018.
- [10] Kelly Street, Davide Risso, Russell B Fletcher, Diya Das, John Ngai, Nir Yosef, Elizabeth Purdom, and Sandrine Dudoit. Slingshot: Cell lineage and pseudotime inference for single-cell transcriptomics. *BMC Genomics*, 19(1):477, 2018.
- [11] Luyi Tian, Shian Su, Daniela Amann-Zalcenstein, Christine Biben, Shalin H Naik, and Matthew E Ritchie. scpipe: a flexible data preprocessing pipeline for single-cell rna-sequencing data. *bioRxiv*, page 175927, 2017.
- [12] Michael B Cole, Davide Risso, Allon Wagner, David DeTomaso, John Ngai, Elizabeth Purdom, Sandrine Dudoit, and Nir Yosef. Performance assessment and selection of normalization procedures for single-cell rna-seq. *bioRxiv*, 2018. doi: 10.1101/235382. URL <https://www.biorxiv.org/content/early/2018/05/18/235382>.
- [13] Jeffrey T Leek, Robert B Scharpf, Héctor Corrada Bravo, David Simcha, Benjamin Langmead, W Evan Johnson, Donald Geman, Keith Baggerly, and Rafael A Irizarry. Tackling the widespread and critical impact of batch effects in high-throughput data. *Nature Reviews Genetics*, 11(10):733, 2010.
- [14] Stephanie C Hicks, F William Townes, Mingxiang Teng, and Rafael A Irizarry. Missing data and technical variability in single-cell rna-sequencing experiments. *Biostatistics*, 2017.
- [15] David van Dijk, Juozas Nainys, Roshan Sharma, Pooja Kathail, Ambrose J Carr, Kevin R Moon, Linas Mazutis, Guy Wolf, Smita Krishnaswamy, and Dana Pe'er. MAGIC: A diffusion-based imputation method reveals gene-gene interactions in single-cell RNA-sequencing data. *bioRxiv*, 2017. doi: 10.1101/111591. URL <http://dx.doi.org/10.1101/111591>.
- [16] Emma Pierson and Christopher Yau. ZIFA: Dimensionality reduction for zero-inflated single-cell gene expression analysis. *Genome Biology*, 16(1):241, dec 2015. ISSN 1474-760X. doi: 10.1186/s13059-015-0805-z. URL <http://genomebiology.com/2015/16/1/241>.
- [17] Koen Van den Berghe, Fanny Perraudeau, Charlotte Soneson, Michael I Love, Davide Risso, Jean-Philippe Vert, Mark D Robinson, Sandrine Dudoit, and Lieven Clement. Observation weights unlock bulk rna-seq tools for zero inflation and single-cell applications. *Genome Biology*, 19(1):24, 2018.

- [18] George C. Tseng and Wing H. Wong. Tight Clustering: A Resampling-Based Approach for Identifying Stable and Tight Patterns in Data. *Biometrics*, 61(1):10–16, mar 2005. ISSN 0006-341X. doi: 10.1111/j.0006-341X.2005.031032.x. URL <http://doi.wiley.com/10.1111/j.0006-341X.2005.031032.x>.
- [19] Trevor Hastie and Robert Tibshirani. Generalized additive models. *Statistical Science*, 1(3):297–318, 1986.
- [20] Aaron TL 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.
- [21] Vladimir Yu Kiselev, Kristina Kirschner, Michael T Schaub, Tallulah Andrews, Andrew Yiu, Tamir Chandra, Kedar N Natarajan, Wolf Reik, Mauricio Barahona, Anthony R Green, et al. Sc3: consensus clustering of single-cell rna-seq data. *Nature methods*, 14(5):483, 2017.
- [22] Rhonda Bacher, Li-Fang Chu, Ning Leng, Audrey P Gasch, James A Thomson, Ron M Stewart, Michael Newton, and Christina Kendziora. Scnorm: robust normalization of single-cell rna-seq data. *Nature methods*, 14(6):584, 2017.
- [23] Cole Trapnell, Davide Cacchiarelli, Jonna Grimsby, Prapti Pokharel, Shuqiang Li, Michael Morse, Niall J Lennon, Kenneth J Livak, Tarjei S Mikkelsen, and John L Rinn. The dynamics and regulators of cell fate decisions are revealed by pseudotemporal ordering of single cells. *Nature biotechnology*, 32(4):381, 2014.
- [24] Zhicheng Ji and Hongkai Ji. Tscan: Pseudo-time reconstruction and evaluation in single-cell rna-seq analysis. *Nucleic acids research*, 44(13):e117–e117, 2016.
- [25] Andrew Butler, Paul Hoffman, Peter Smibert, Efthymia Papalexis, and Rahul Satija. Integrating single-cell transcriptomic data across different conditions, technologies, and species. *Nature Biotechnology*, 2018. ISSN 1087-0156. doi: 10.1038/nbt.4096. URL <https://www.nature.com/articles/nbt.4096>.
- [26] Peter V Kharchenko, Lev Silberstein, and David T Scadden. Bayesian approach to single-cell differential expression analysis. *Nature methods*, 11(7):740, 2014.
- [27] Luke Zappia, Belinda Phipson, and Alicia Oshlack. Exploring the single-cell rna-seq analysis landscape with the scran-tools database. *PLOS Computational Biology*, 14(6):e1006245, 2018.