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

Fanny Perraudeau[1], Davide Risso[2], Kelly Street[1], Elizabeth Purdom[3], and Sandrine Dudoit[4]

[1]**Graduate Group in Biostatistics, University of California, Berkeley, Berkeley, CA**
[2]**Division of Biostatistics and Epidemiology, Department of Healthcare Policy and Research, Weill Cornell Medicine, New York, NY**
[3]**Department of Statistics, University of California, Berkeley, Berkeley, CA**
[4]**Division 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 unprecendented 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

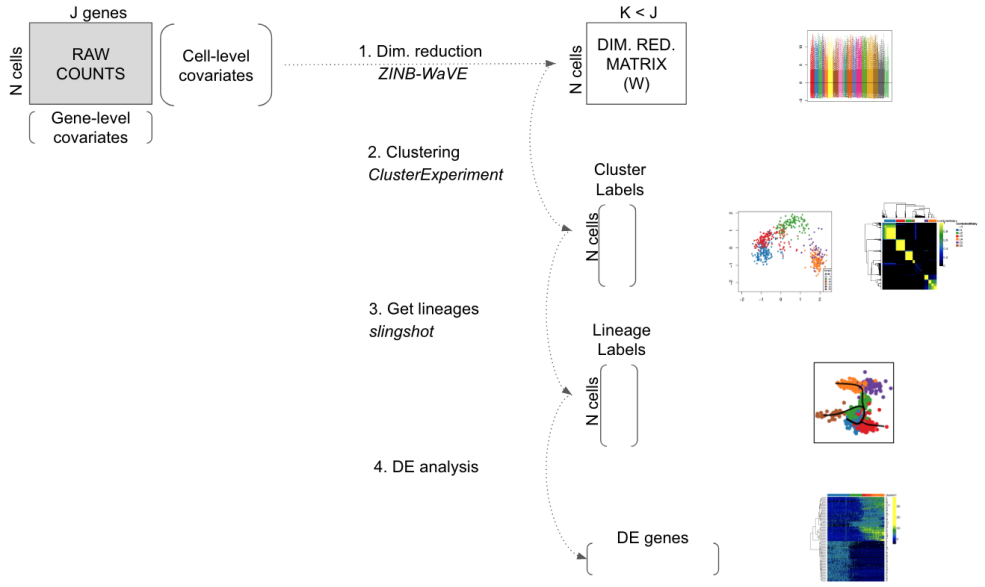

Figure 1. Workflow for analyzing scRNA-seq datasets. On the right, main plots generated by the workflow.

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 pseudotimes, 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 `SCESet` class allowing integrated analysis. 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 dowstream 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` R package; and (4) DE analysis along lineages. Throughout the workflow, we use a single `SingleCellExperiment` object to store the scRNA-seq data along with any gene or cell-level metadata available from the experiment.

EP: I have only updated the code, not the text. The text needs adaptation, especially future directions and references to `SummarizedExperiment` which should be `SingleCellExperiment`. FP: TODO, fix ref.

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) [4]. The olfactory epithelium contains mature olfactory sensory neurons (mOSN) that 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.

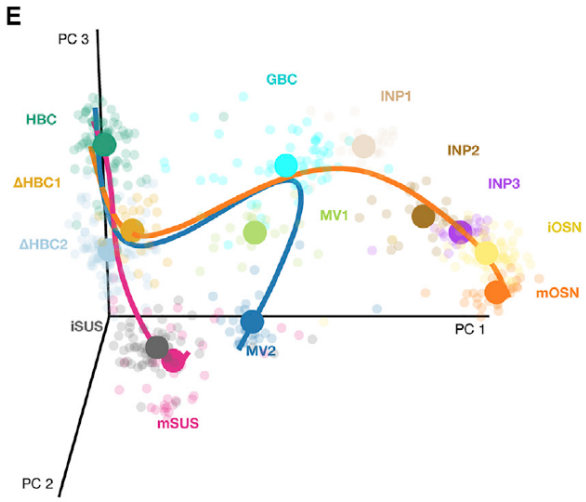

Figure 2. Stem cell differentiation in the mouse olfactory epithelium. This figure was reproduced with kind permission from Fletcher et al. (2017).

The scRNA-seq dataset we use as a case study was generated to study the differentitation 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 [4, 5, 6].

It was found that the first major bifurcation in the HBC lineage trajectory occurs prior to cell division, producing either mature sustentacular (mSUS) cells or GBCs. Then, the GBC lineage, in turn, branches off to give rise to mOSN, microvillous (MV) cells, and cells of the Bowman gland (Figure 2). 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`.

Package versions

The following packages are needed.

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

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

set.seed(20)
```

Note that in order to successfully run the workflow, we need the devel 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/`).

EP: Should update these.

DR: Done, but let's double-check again before the final submission.

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 2 cores on a MacBook Pro (early 2015) with a 2.7 GHz Intel Core i5 processor and 8 GB of RAM. The `BiocParallel` package is used to allow for parallel computing in `zinbwave` function. Users with a different operating system may change the package used for parallel computing and the `NCORES` variable below.

```
NCORES <- 2
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())
}
```

EP: Note that clusterExperiment uses `parallel` package, so might should update this.

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%5FoeHBCd
         "https://raw.githubusercontent.com/rufletch/p63-HBC-diff/master/ref/oeHBCdiff_clusterLabels

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 [4] for details).

EP: There's code in SingleCellExperiment for spike-ins. We should use it, though I'm not familiar with it. FP: Davide, what is the standard way to use SingleCellExperiment for spike-ins?

```
# 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
```

Throughout the workflow, we use the class `SingleCellExperiment` to keep track of the counts and their associated metadata within a single object. The cell-level metadata contain quality control measures, sequencing batch ID, and cluster and lineage labels from the original publication [4]. 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

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

```
## 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`, 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,
```

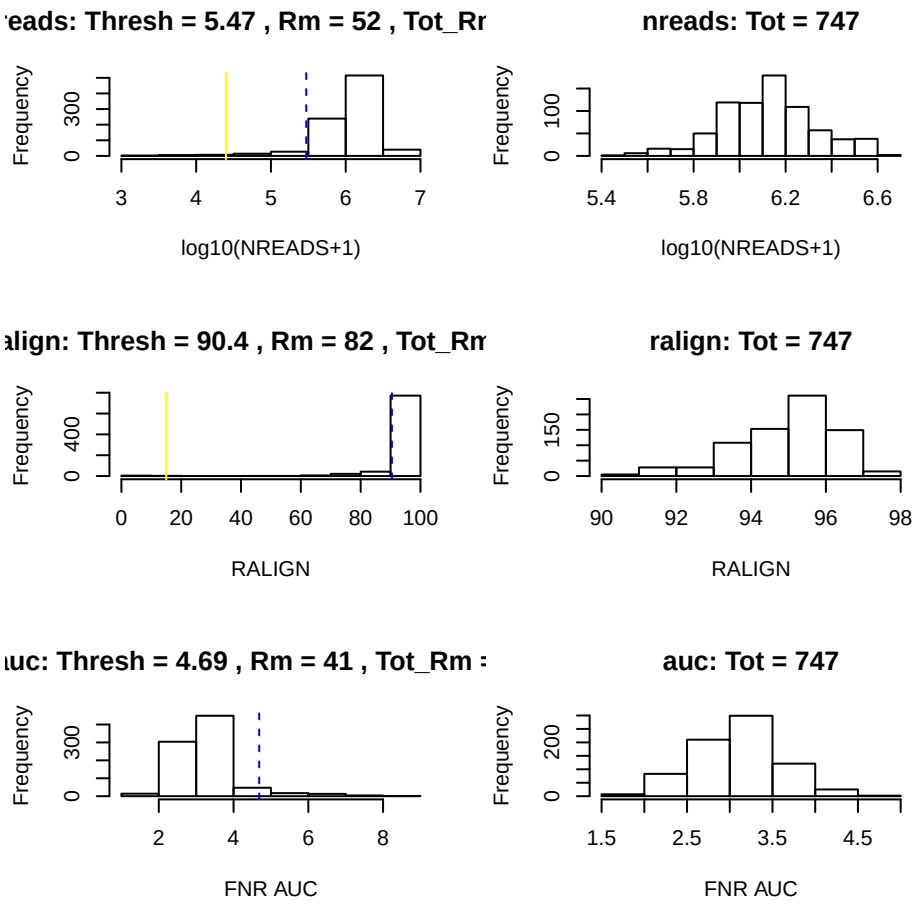

Figure 3. SCONE: Filtering of low-quality cells.

```
                              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.

Finally, for computational efficiency, we retain only the 1,000 most variable genes. This seems to be a reasonnable choice for the illustrative purpose of this workflow, as we are able to recover the biological signal found in the published analysis ([4]). In general, however, we recommend care in selecting a gene filtering scheme, as an appropriate choice is dataset-dependent.

```
# # Filtering to top 1,000 most variable genes
vars <- rowVars(log1p(counts(dataObj)))
names(vars) <- rownames(dataObj)
vars <- sort(vars, decreasing = TRUE)
# dataObj <- dataObj[names(vars)[1:1000],]

#alternative script:
dataObj<-makeFilterStats(dataObj,filterStats="var", transFun = log1p)
dataObj<-filterData(dataObj,percentile=1000,filterStats="var")

length(intersect(rownames(dataObj), names(vars)[1:1000]))
```

```
## [1] 1000
```

```
setdiff(rownames(dataObj), names(vars)[1:1000])
```

```
## character(0)
```

EP: Added existing code in the clusterExperiment package that does this. Also will mean that any future steps in package that need variance for the dataObj will have variance per gene stored.
FP: great. I added the filterStats argument to filterData function. I get an error otherwise.

FP: We get a different list of most variable genes here.

DR: Fixed, the new code was computing the variance in linear space instead of using the log.

DR: Remember to remove the old code and the length() and setdiff() from above.

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 batches.

```
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 [4], 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
```

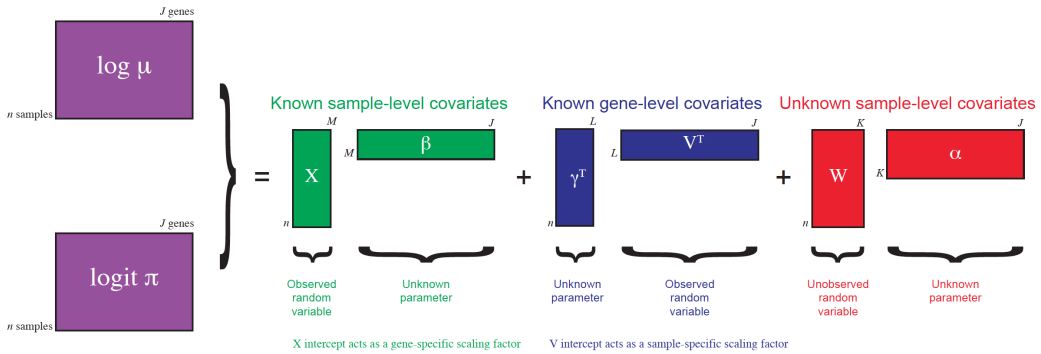

Figure 4. ZINB-WaVE: Schematic view of the ZINB-WaVE model. This figure was reproduced with kind permission from Risso et al. (2017).

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  0  2  0  2  9
##   GBC08B  8  0  7  5  3  0  0  0  1  2  3  0  5  6
##   GBC09A  6  0  1  5  8  0  0  0  1  1  0  0  6  7
##   GBC09B 12  0  2  1  3  0  0  0  1  0  0  0  3  0
##   P01     7  0  2  4  3 15  0  0  0  0  0  0  0  0
##   P02     5  2  0  9  3 15  3  3  2  3  0  2  1  0
##   P03A   15  3  0  2  0 12  2  9  4  2  0  2  0  0
##   P03B    9  1  2  1  1 11  1  2  8  1  1  2  0  0
##   P04     8  0  0  0  0  9  1  0  1  1  0  0  0  0
##   P05     3  0  0  0  1 11  3  0  1  0  2  2  0  0
##   P06    12  1  2  3  0  8  2  4  8  4  1  2  2  2
##   P10     7  3  1  4  0  3  5  8  1  0  2  5  0  1
##   P11     6  2  1  1  0  1  5  1 22  3  1  6  0  1
##   P12    10  0  2  0  0  4 10  0  8  2  3  6  4  1
##   P13    13  1  2  4  0  4 15  0  4  5  6  1  3  2
##   P14     9  0  0  1  2  0 11  0 12  2  0  7  0  3
##   Y01     8 46  1  1  2  0  0  0  0  0  0  0  0  0
##   Y04    10 31  0  1  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 [7, 8, 5].

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 [5].

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.

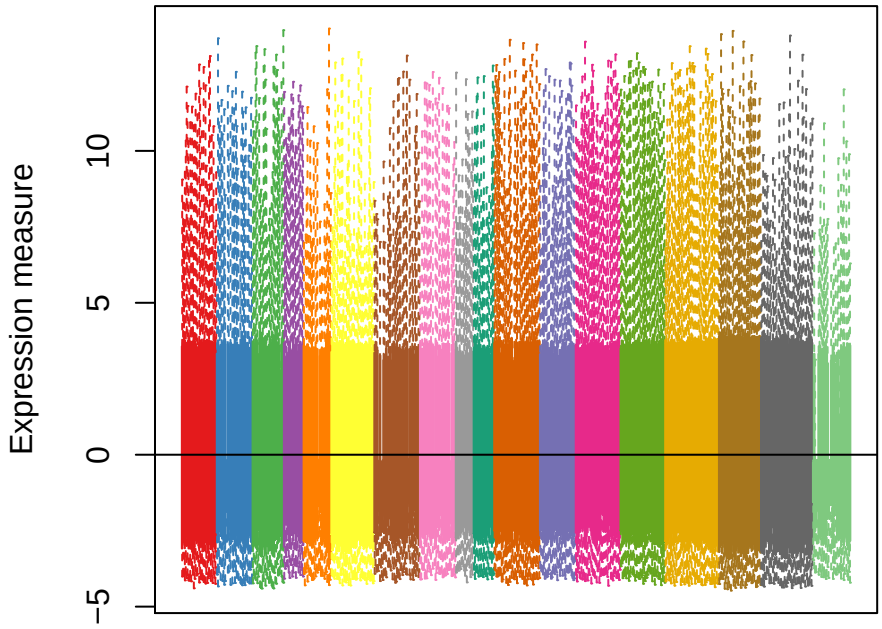

Figure 5. ZINB-WaVE: Boxplots of normalized expression measures (deviance residuals), color-coded by batch.

```
print(system.time(dataObj <- zinbwave(dataObj, K = 50, X = "~ Batch", residuals = TRUE, normalizedVa
```

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

Normalization

The function `zinbwave` returns a `SummarizedExperiment` object that includes normalized expression measures, 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          4.531898        4.369185       -4.142982
## Cyp2f2        4.359680        4.324476        4.124527
## Gstm1         4.724216        4.621898        4.403587
```

FP: These values are slightly different from what

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)
```

The principal component analysis (PCA) of the normalized values shows that, as expected, cells do not cluster by batch but by the original clusters (Figure 6). Overall, it seems that normalization was effective at removing batch effects without removing biological signal, in spite of the partial nesting of batches within clusters.

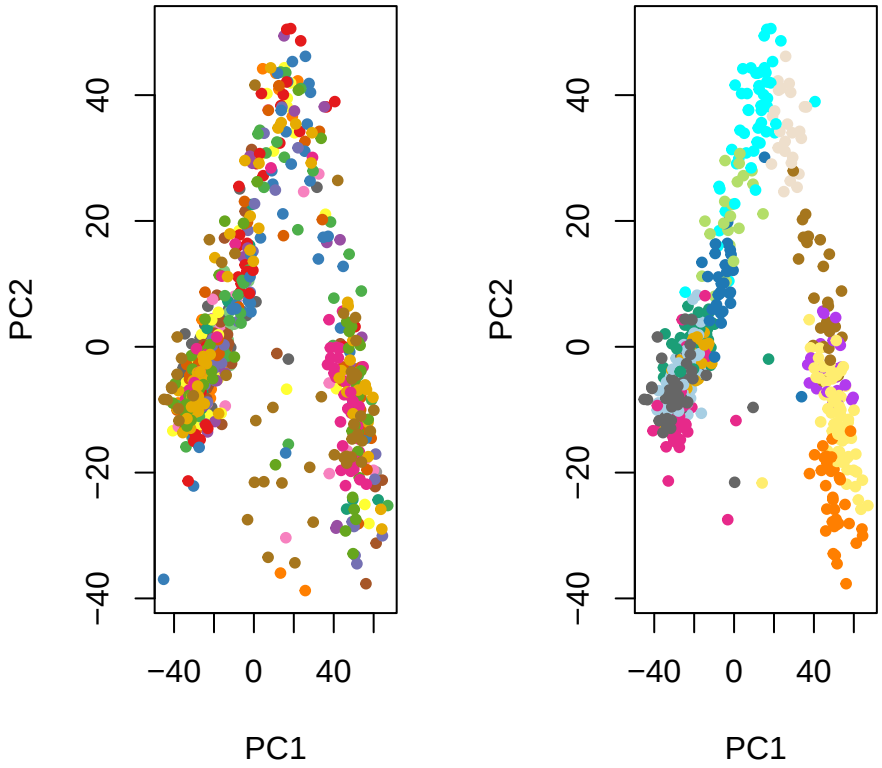

Figure 6. ZINB-WaVE: PCA of normalized expression measures, where each point represents a cell. Cells are color-coded by batch (left panel) and by original published clustering (right panel).

```
pca <- prcomp(t(norm))
par(mfrow = c(1,2))
plot(pca$x, col = col_batch[batch], pch = 20, main = "")
plot(pca$x, col = col_clus[as.character(publishedClusters)], pch = 20, main = "")
```

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")]
```

EP: This saves a lot of trouble with the plotHeatmap, etc. FP: Ok.

EP: I've changed this code from my previous ad hoc to not delete the previous, but just reorder them. Again, (it might make sense for me to add option to different assays in clusterExperiment, but I'm not going to do that right now.)

EP: Should probably address these normalized values, especially to more strongly make clear that shouldn't use these for clustering, just for visualization. The way its stated it sounds like dimensionality reduction is an "also", when in fact it is the main point and the normalized values are the afterthought. Might consider reordering the sections to more clearly demonstrate this and put dimensionality reduction first.
FP: Agree, let's keep it in mind for once the workflow runs smoothly.

DR: Agree, I'm not even sure normalized values are that useful for visualization... a simple scaling by total counts would probably achieve the same... but since we have them...

Dimensionality reduction

The `zinbwave` function can also be used to perform dimensionality reduction, where, 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 7).

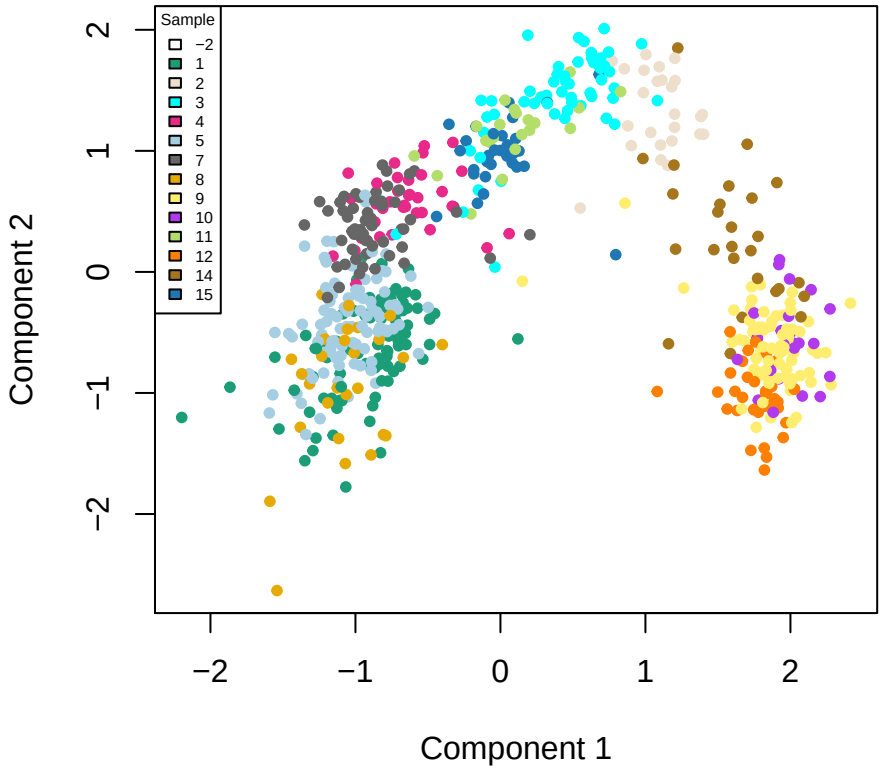

Figure 7. ZINB-WaVE: MDS of the low-dimensional matrix W, where each point represents a cell and cells are color-coded by original published clustering.

```
W <- reducedDim(dataObj)
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 = "
```

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 [9]. 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.

Note that the defaults in `RSEC` are designed for input data that are the actual (normalized) counts. Here, we are applying `RSEC` instead to the low-dimensional `W` matrix from ZINB-WaVE, for which we make a separate `SummarizedExperiment` object. For this reason, we choose to not use certain options in `RSEC`. In particular, we do not use the default dimensionality reduction step, since our input `W` is already in a space of reduced dimension. Specifically, `RSEC` offers a dimensionality reduction option for the input to both the clustering routines (`reduceMethod`) and the construction of the hiearchy between the clusters (`dendroReduce`). We also skip the option to merge our clusters based on the amount of differential gene expression between clusters.

FP: As the name/type “zinbwave50” is then needed in the RSEC function, we might need to keep this line. EP: I just called it “zinbwave50” for convenience. I’ve changed future code to just use “zinbwave”. Though as a suggestion, you might consider for the zinbwave function to add the value of K to the name automatically,

since the dimensions are not nested within each other, like pca, so you might run multiple versions of zinbwave and want to compare different K.

EP: Note that we need to see whether clusterExperiment retains all of the different assays or not. It should in principle, but its not something I've tested. Also, I purposefully didn't give the parameter `nReducedDims` in RSEC because I think it should automatically use the max saved in object if not designated. If we're not getting the same results, we can add in `nReducedDims=50` and see; if that changes anything, let me know. However, I would focus on the question of different W values.

FP: I am not sure it makes sense to perform the DE on the normalized values. Intuitively, I would say no, but not sure. We can still show that it is a possibility and write in the text that in this case, it might not make sense. We are working on a DE method for scRNA-seq where we use observational weights computed from the posterior probabilities of the ZINB-WaVE model in bulk RNA tools like edgeR and DESeq2. I think, eventually, it would be nice to be able to use that. Maybe, not for the next next version of the workflow.
EP: I'd note that we use the normalized values for finding those DE with respsect to lineage. But I don't think its necessary to put it in here, but maybe in future directions, where you discuss that functionality, we could mention that it would allow the usage of mergeClusters. Also would be a good time to point out that DE on normalized values probably not a great idea either, if you don't think so. I also think the reason for skipping mergeClusters should be stated in the above text, rather than just saying we are skipping it.

DR: A lot of these comments are now outdated, as the new version of clusterExperiment fully support multiple assays. Similarly, the weights are now a reality and also supported by clusterExperiment. We should probably add a section about it?

```
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)))
```

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

The resulting candidate clusterings can be visualized using the `plotClusters` function (Figure 8), 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 accross rows. The first row correspond to a consensus clustering across all candidate clusterings.

```
plotClusters(dataObj)
```

FP: We get more clusters for the makeConsensus method than before. Maybe because the most variable genes are different?

DR: I fixed the most variable genes. Is there a way to check the older results? Where do I find the saved objects?

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 9).

```
plotCoClustering(dataObj)
```

FP: Ordering does not seem right.
EP: I get a very different plotCoClustering plot than what you sent me. See email.

The distribution of cells across the consensus clusters can be visualized in Figure 10 and is as follows:

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

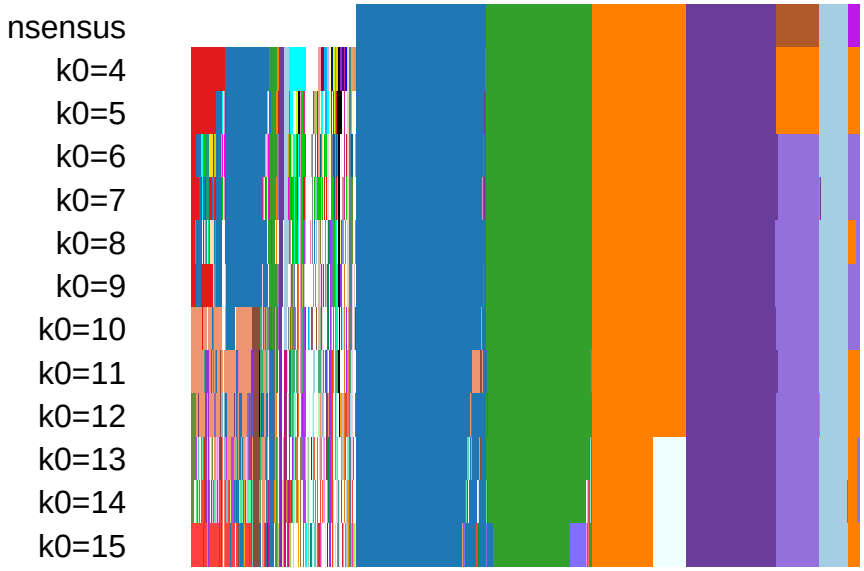

Figure 8. **RSEC: Candidate clusterings found using the function RSEC from the clusterExperiment package.**

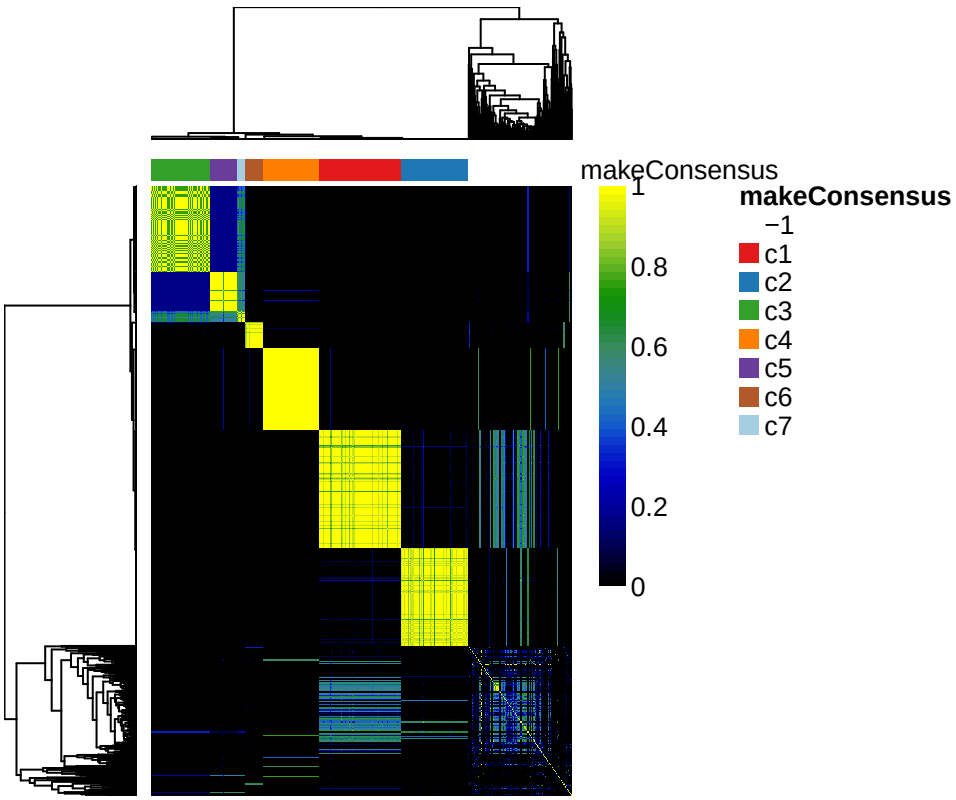

Figure 9. **RSEC: Heatmap of co-clustering matrix.**

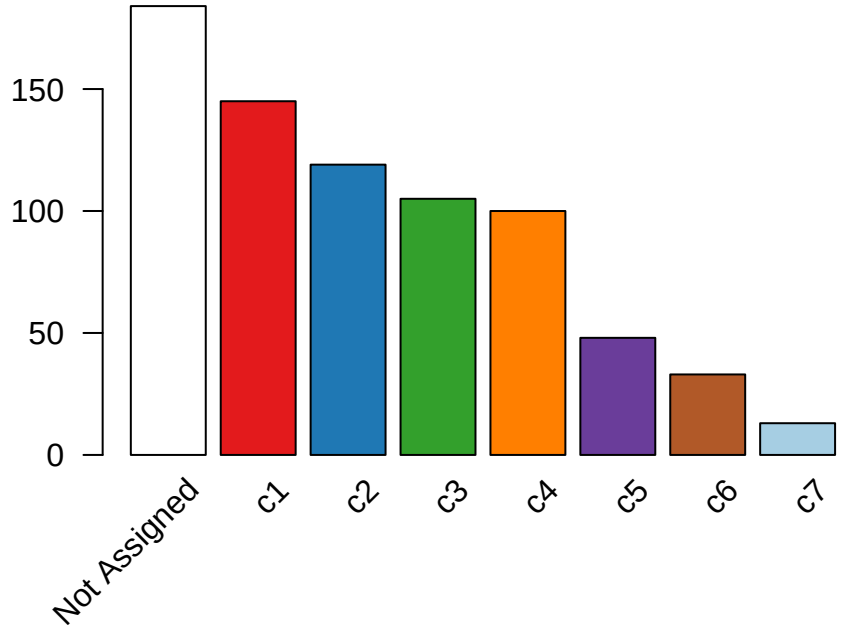

Figure 10. RSEC: Barplot of number of cells per cluster for our workflow's RSEC clustering.

```
##
##  -1  c1  c2  c3  c4  c5  c6  c7
## 184 145 119 105 100  48  33  13
```

FP: we do not get exactly the same number of cells in each cluster. It looks similar though.

DR: it's pretty similar, but we still get 7 clusters that look more or less the same as before.

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

The distribution of cells in our workflow's clustering overall agrees with that in the original published clustering (Figure 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.

```
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")
```

FP: compared to the previous version, it seems that there is a lot of black. Would the black coming from the contour of the rectangles in the bars? EP: I'm not sure what you mean about a lot of black on the barplot. Do you mean the unassigned samples? The colors have changed because the default set of colors was updated.

DR: Now it looks pretty much the same.

DR: Better to use `plotClustersTable` here? See below.

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

Figure 13 displays a heatmap of the normalized expression measures for the 1,000 most variable genes, where cells are clustered according to the RSEC consensus.

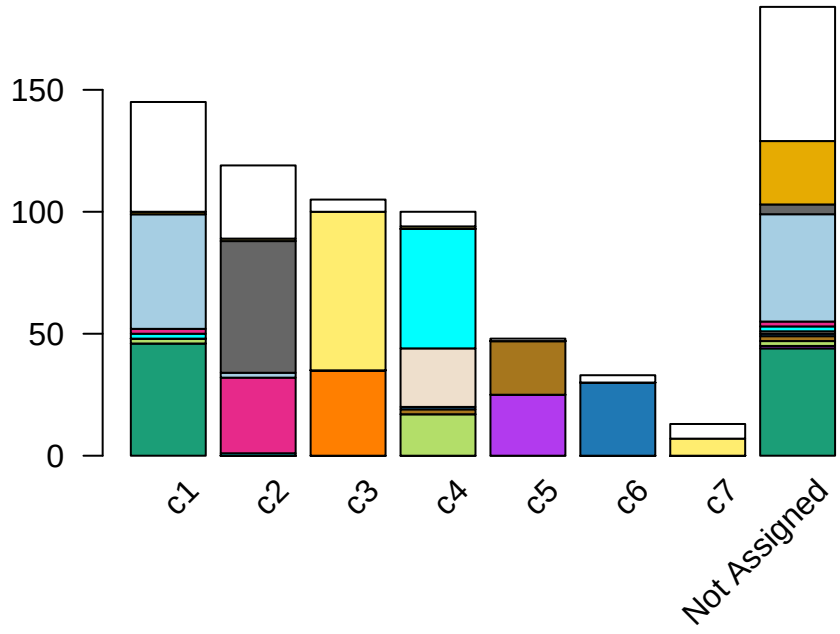

Figure 11. RSEC: Barplot of number of cells per cluster, for our workflow's RSEC clustering, color-coded by original published clustering.

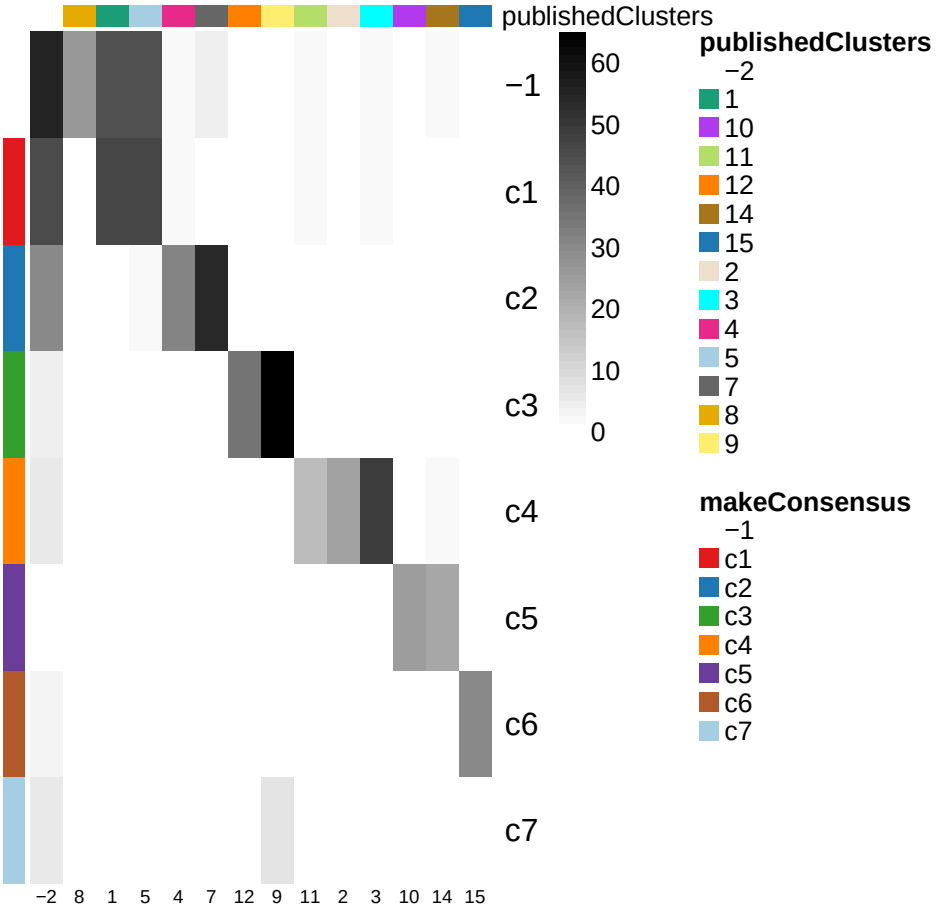

Figure 12. RSEC: Confusion matrix of number of cells per cluster, for our workflow's RSEC clustering and the original published clustering.

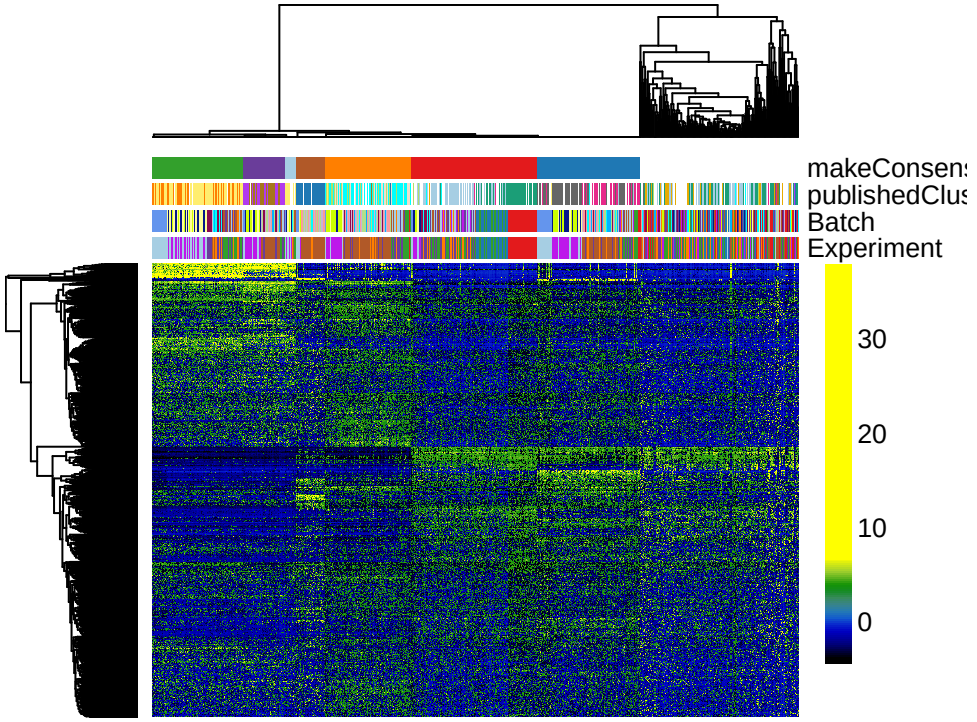

Figure 13. RSEC: Heatmap of the normalized expression measures for the 1,000 most variable genes, where rows correspond to genes and columns to cells ordered by RSEC clusters.

```
# 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,
            sampleData = c("Batch", "Experiment"),
            clusterLegend = metaColors, annLegend = FALSE, main = "")
```

FP: publishedClusters does not appear in the bars. EP: I forgot to add it to the `whichClusters` argument.

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 14); this is anologous to Figure 7 for the original published clusters.

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

```
# ###test
# col<-convertClusterLegend(dataObj,whichCluster="primary",output="matrixColors")[,1]
# col[col=="white"]<-"grey"
# W<-reducedDim(dataObj,"zinbwave")
# plot(-W[,1:2],col=col,pch=20)
```

FP: I get `"reduceMethod" is not a graphical parameter`. I think this plot is different from the previous version.

EP: I had wrong name for argument which has changed (should have been `reducedDim`), so it went with default which is PCA on the assay(dataObj) slot – i.e. not using zinbwave. Now it gives the same plot as my test, where I directly plotted the results of `reducedDim(dataObj,"zinbwave")`. However, its still not

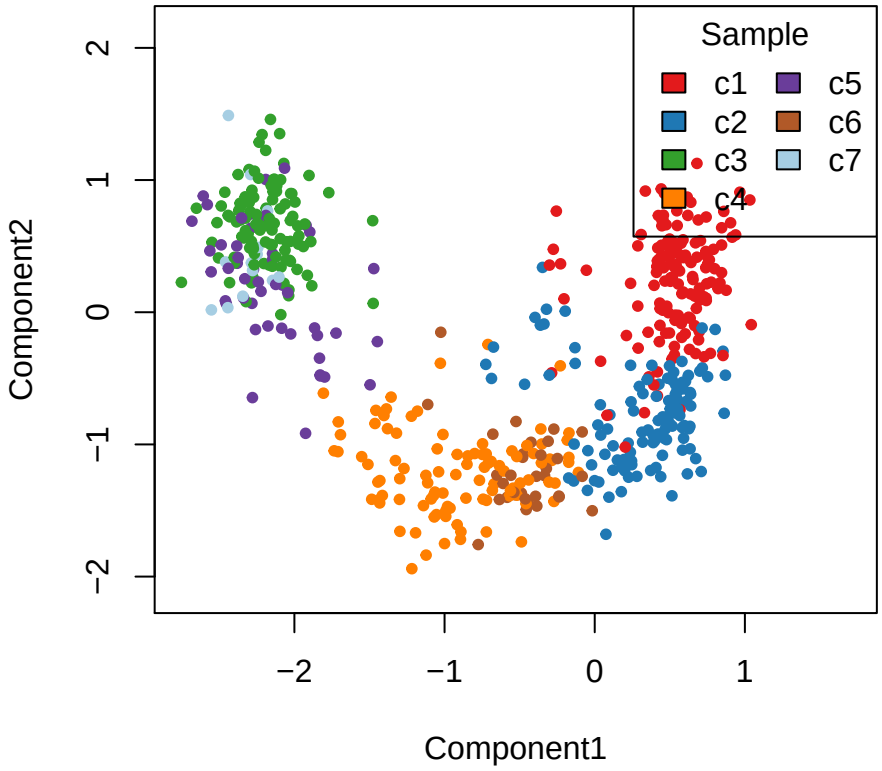

Figure 14. RSEC: MDS of the low-dimensional matrix W, where each point represents a cell and cells are color-coded by RSEC clustering.

exactly the same as previous rendition online. First it looks like the dimensions got flipped, i.e. multiplied by -1. Furthermore, it doesn't look exactly the same (e.g. outlying points not the same), even if I color in the unassigned so we can see them. Is zinbwave giving back the exact same W as before? I've checked that the W in the `dataObj_afterZinbwave` and W I plotted above from `dataObj_afterRSEC` are the same so its not clusterExperiment messing something up. If so, this could be why we are getting slightly different clustering results.

DR: It is possible that zinbwave does not return the same exact W. There is no guarantee of a global maximum, but only local. The plot looks more or less the same, but just flipped.

Cell lineage and pseudotime inference: Slingshot

We now demonstrate how to use the R software package `slingshot` to infer branching cell lineages and order cells by developmental progression along each lineage. The method, proposed in [6], 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.

The two steps of the Slingshot algorithm are implemented in the functions `getLineages` and `getCurves`, respectively. The first takes as input a low-dimensional representation of the cells and a vector of cluster labels. It fits an MST to the clusters and identifies lineages as paths through this tree. The output of `getLineages` is an object of class `SlingshotDataSet` containing all the information used to fit the tree and identify lineages. The function `getCurves` then takes this object as input and fits simultaneous principal curves to the identified lineages. These functions can be run separately, as below, or jointly by the wrapper function `slingshot`.

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). 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 c2 c3 c4 c5 c6 c7
##       -2 55 45 30  5  6  1  3  6
##       1  44 46  0  0  0  0  0  0
##       2   1  0  0  0 24  0  0  0
##       3   2  2  1  0 49  0  0  0
##       4   2  2 31  0  0  0  0  0
##       5  44 47  2  0  0  0  0  0
##       7   4  0 54  0  0  0  0  0
##       8  26  1  0  0  0  0  0  0
##       9   0  0  1 65  1  0  0  7
##       10  1  0  0  0  0 25  0  0
##       11  2  2  0  0 17  0  0  0
##       12  0  0  0 35  0  0  0  0
##       14  2  0  0  0  2 22  0  0
##       15  1  0  0  0  1  0 30  0
```

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

Cells in cluster `c4` have a cluster label of `-2` in the original published clustering, meaning that they were not assigned to any cluster. These cells were actually identified as non-sensory contaminants, as they overexpress gene `Reg3g` (see Figure S1 from [4] and Figure 15), and were removed from the original published clustering. While it is reassuring that our workflow clustered these cells separately, with no influence on the clustering of the other cells, we removed cluster `c4` to infer lineages and pseudotimes, as cells in this cluster do not participate in the cell differentiation process. Note that, out of the 77 cells overexpressing `Reg3g`, 11 are captured in cluster `c4` and 21 are unclustered in our workflow's clustering (see Figure 15). However, we retain the remaining 45 cells to infer lineages as they did not seem to influence the clustering.

DR: This has changed (slightly). This is the table that I get currently.

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

```
plotFeatureBoxplot(dataObj, whichCluster="primary", feature="Reg3g",ylab = "Reg3g count values", cex
```

FP: this plot is fairly different from the previous version. We should see for one cluster that the gene expression for Reg3g is much high that for the other clusters.

EP: I think this plot showed the normalized values (saved in assay(dataObj)). But I see that the previous plot was on the counts. Is that better than normalized value? I have added (just to this plot for now), an option to plot a specific given matrix (or a character value defining an assay, but since we want log-transform of the counts, which is not saved in the main assay, putting in "counts" doesn't give plot we want). But it's now cluster c1, not c4 that has this property. It might be useful to take the clustering from the old RSEC data object and directly compare the clusterings. Regardless, none of the previous commands are going to get the right results going forward since the clusters don't have the same name. See note on plotReducedDims command where I think the input `W` has changed.

DR: I think we should re-write this section. With the current version of the package and parameters, we don't get the contaminant cluster anymore, but most of the contaminant cells are now in c1. However c1 also includes HBC cells, so we should not remove it from the slingshot analysis.

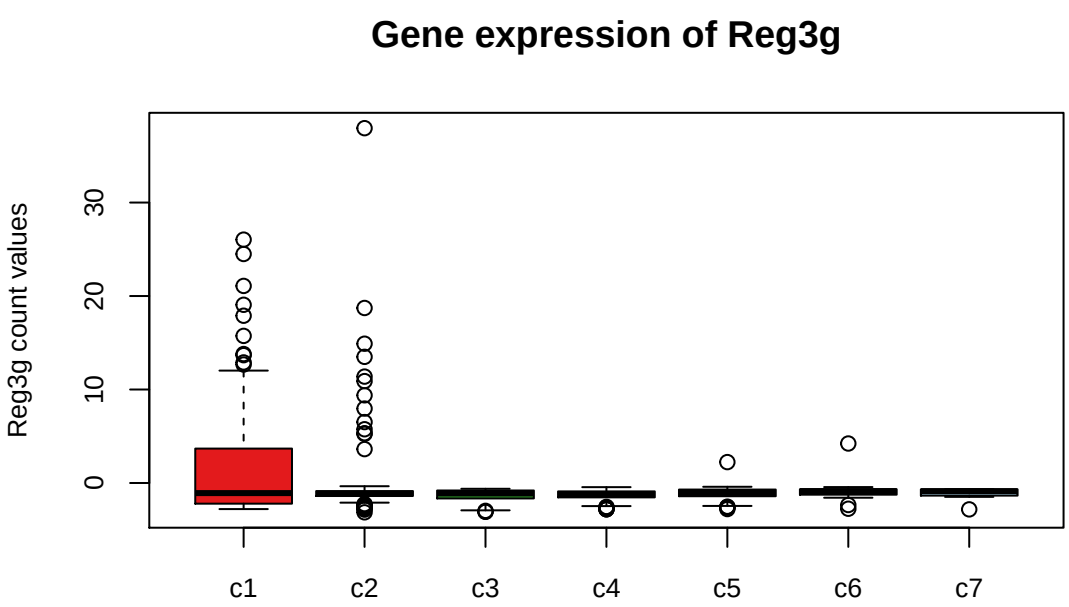

Figure 15. RSEC: Boxplots of the log count of gene Reg3g stratified by cluster.

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

EP: See my email. I don't get this subsetting error on my machine or on TravisCI. I do get on SCF Servers which are running R3.4.1 and the older version of bioconductor. Can you check this on your machine?

FP: I agree. I get an error though when I run line 501: `Error in .BasicFunsList[[f]] : no such index at level 1`.

FP: I stooped here.

DR: Above, I use the syntax to work directly on the ClusterExperiment object, but if we want to be able to plot the resulting lineages, we need the matrix method.

Before fitting 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 work (Figure 16). The `slingshot` package also includes functionality for 3-dimensional visualization as in Figure 2, using the `plot3d` function from the package `rgl`.

```
colCl<-convertClusterLegend(pseudoCe,whichCluster="primary",output="matrixColors")[,1]
lineages <- getLineages(mds$points, clusterLabels = primaryClusterNamed(pseudoCe), start.clus = "c1"
pairs(lineages, type="lineages", col = colCl)
```

DR: For some reason, this plot does not work anymore, but the lineages that I get are correct!

Having found the global lineage structure, we now construct a set of smooth, branching curves in order to infer the pseudotime variables. Simultaneous principal curves are constructed from the individual cells along each lineage, rather than the cell clusters. This makes them more stable and better suited for assigning cells to lineages. The final curves are shown in Figure 17.

```
lineages <- getCurves(lineages)
pairs(lineages, type="curves", col = colCl)
```

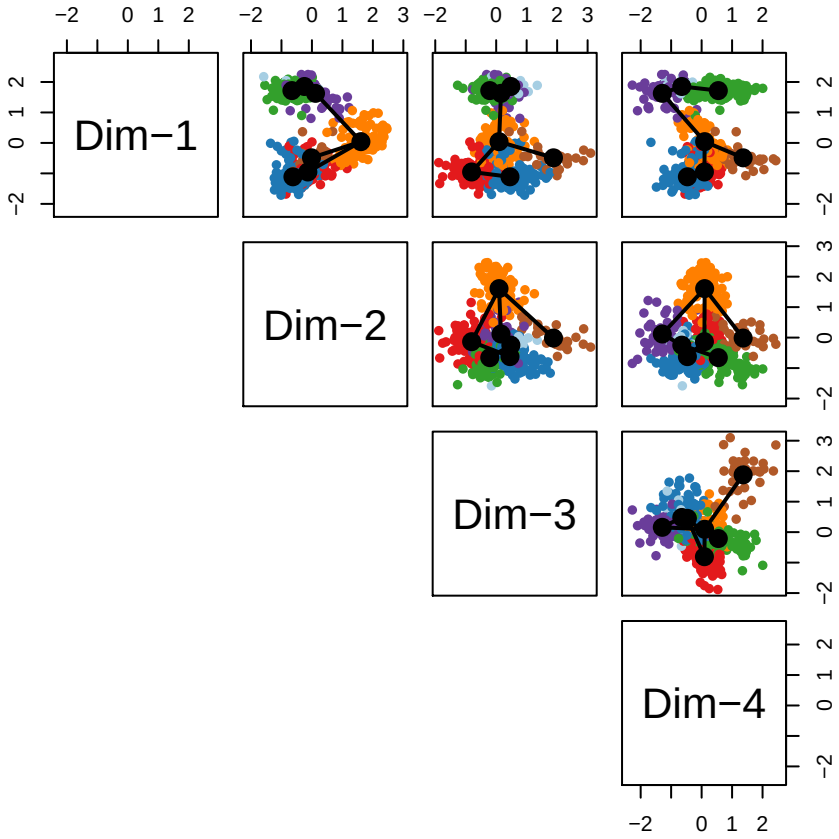

Figure 16. Slingshot: Cells color-coded by cluster in a 4-dimensional MDS space, with connecting lines between cluster centers representing the inferred global lineage structure.

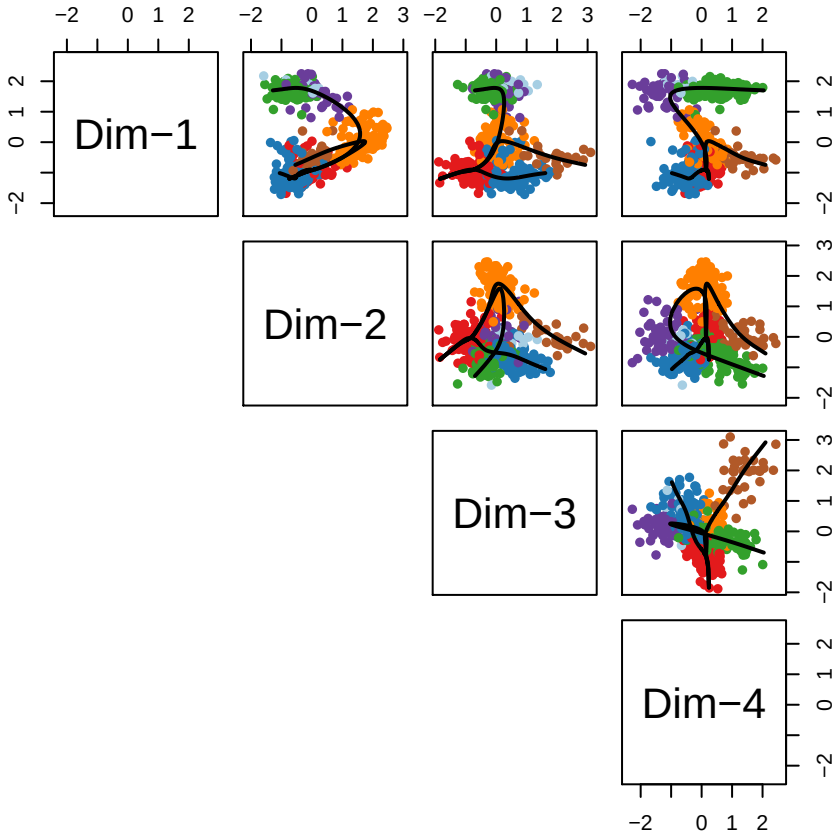

Figure 17. Slingshot: Cells color-coded by cluster in a 4-dimensional MDS space, with smooth curves representing each inferred lineage.

```
lineages
```

```
## class: SlingshotDataSet
##
##  Samples Dimensions
##      563          4
##
## lineages: 3
## Lineage1: c1  c4  c5  c7  c3
## Lineage2: c1  c4  c6
## Lineage3: c1  c2
##
## curves: 3
## Curve1: Length: 9.5224   Samples: 361.18
## Curve2: Length: 7.8206   Samples: 234.01
## Curve3: Length: 4.2824   Samples: 254.86
```

In the workflow, we recover a reasonable 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. `getLineages` 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.

```
lineages <- getLineages(X, clusterLabels = primaryClusterNamed(pseudoCe), start.clus = "c1",
                        end.clus = c("c3", "c7"))
lineages <- getCurves(lineages)
pairs(lineages, type="curves", col = colorCl)
pairs(lineages, type="lineages", col = colorCl,
      show.constraints = TRUE)

lineages
```

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. 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 18.

```
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
})
```

EP: Since changed NormalizedValues to the main assay, this cleans up the code above and below. I've used the `transformData` command, which will not do anything if isCount=FALSE above, but will convert to right if we change isCount=TRUE. It's better to use that command than `assay` for that reason. Might should add comment in code or text explaining this.

DR: Kelly, can you please check why I get different lineages and pseudotimes for the slingshot method and for the getLineages/getCurves method? The latter seems the correct one.

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

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

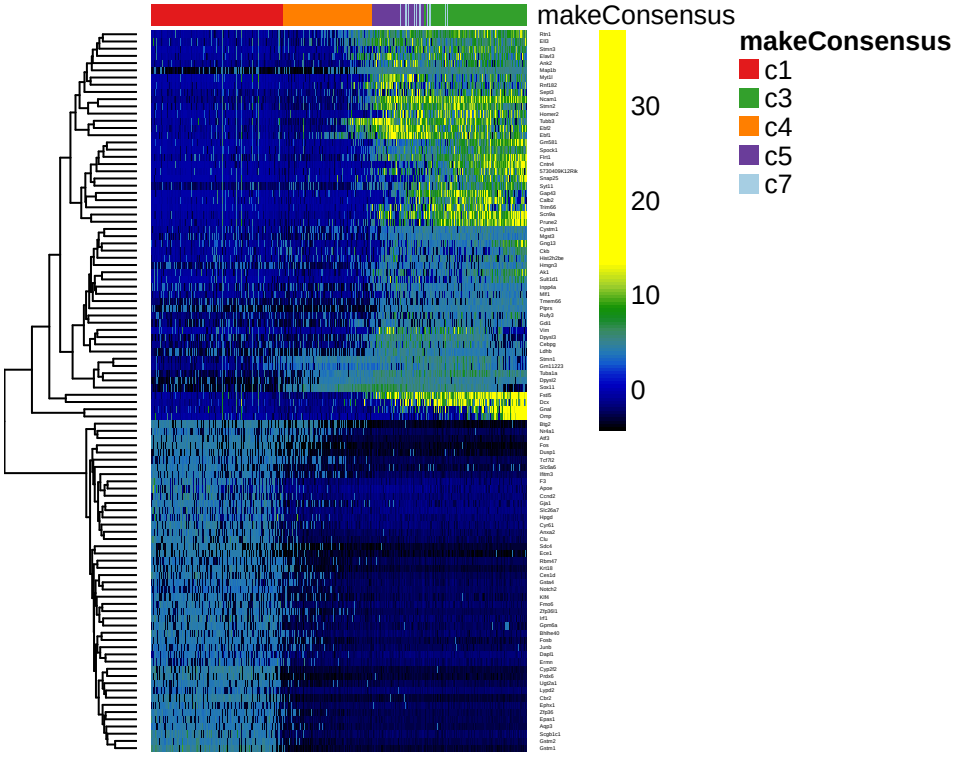

Figure 18. DE: Heatmap of the normalized expression measures for the 100 most significantly DE genes for the neuronal lineage, where rows correspond to genes and columns to cells ordered by pseudotime.

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

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

EP: note the above does heirarchcial clustering of the top genes. If want them to stay in order given, should give `clusterFeaturesData=topgenes` (I think that is the argument)

EP: The above code should take care of the ordering and NA values. Note that subseting a ClusterExperiment object is suppose to also subset the orderSamples value correctly. Let me know if that is not the case.

Further developments

In an effort to improve scRNA-seq data analysis workflows, we are currently exploring a variety of applications and extensions of our ZINB-WaVE model. In particular, we are developing a method to impute counts for dropouts; the imputed counts could be used in subsequent steps of the workflow, including dimensionality reduction, clustering, and cell lineage inference. In addition, we are extending ZINB-WaVE to identify differentially expressed genes, both in terms of the negative binomial mean and the zero inflation probability, reflecting, respectively, gradual DE and on/off DE patterns. We are also developing a method to identify genes that are DE either within or between lineages inferred from Slingshot.

Finally, a new S4 class called `SingleCellExperiment` is currently under development (`https://github.com/drisso/SingleCellExperiment`). This new class is essentially a `SummarizedExperiment` class with a couple of additional slots, the most important of which is `reducedDims`, which, much like the `assays` slot of `SummarizedExperiment`, can contain one or more matrices of reduced dimension. This new `SingleCellExperiment` class would be a valuable addition to the workflow, as we could store in a single object the raw counts as well as the low-dimensional matrix created by the ZINB-WaVE dimensionality reduction step. Once the implementation of this class is stable, we would like to incorporate it to the workflow.

EP: Obviously should replace this text. Might want to mention ability of class to store data matrices not in memory, though not currently implemented, which is the next major change we need to do here.

Conclusion

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 sustitute 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.

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://github.com/kstreet13/slingshot`.

```
sessionInfo()
```

```
## R version 3.5.0 (2018-04-23)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS High Sierra 10.13.5
##
## 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_1.0.9
##  [5] foreach_1.4.4               slingshot_0.99.9
##  [7] princurve_2.0.2             zinbwave_1.3.2
##  [9] scone_1.5.0                 clusterExperiment_2.1.5
## [11] bigmemory_4.5.33            SingleCellExperiment_1.3.6
## [13] SummarizedExperiment_1.11.5 DelayedArray_0.7.16
## [15] matrixStats_0.53.1          Biobase_2.41.1
## [17] GenomicRanges_1.33.6        GenomeInfoDb_1.17.1
## [19] IRanges_2.15.14             S4Vectors_0.19.16
## [21] BiocGenerics_0.27.1         BiocParallel_1.15.6
## [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       RNeXML_2.1.1
```

```
##   [9] devtools_1.13.5          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_0.21.0             GenomeInfoDbData_1.1.0
##  [25] hwriter_1.3.2            bit64_0.9-7
##  [27] rhdf5_2.25.4             rprojroot_1.3-2
##  [29] xfun_0.2                 EDASeq_2.15.2
##  [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_1.15.0            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_1.4.4.1           tensorA_0.36
##  [57] tools_3.5.0              bookdown_0.7
##  [59] gridBase_0.4-7           ggplot2_2.2.1.9000
##  [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] purrr_0.2.5              RCurl_1.95-4.10
##  [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_3.10.0
##  [79] mime_0.5                 hms_0.4.2
##  [81] evaluate_0.10.1          xtable_1.8-2
##  [83] XML_3.98-1.11            mclust_5.4
##  [85] gridExtra_2.3            compiler_3.5.0
##  [87] biomaRt_2.37.1           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_1.7.1
##  [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
## [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_1.17.2 registry_0.5
## [117] numDeriv_2016.8-1        locfdr_1.1-8
## [119] xml2_1.2.0               rARPACK_0.11-0
## [121] annotate_1.59.0          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_1.33.0
```

```
## [139] gtools_3.8.1             modeltools_0.2-21
## [141] jsonlite_1.5             nlme_3.1-137
## [143] bindrcpp_0.2.2           Rhdf5lib_1.3.1
## [145] viridisLite_0.3.0        limma_3.37.2
## [147] pillar_1.2.3             lattice_0.20-35
## [149] httr_1.3.1               DEoptimR_1.0-8
## [151] survival_2.42-3          glue_1.2.0
## [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_1.9.3          blob_1.1.1
## [161] latticeExtra_0.6-28      caTools_1.17.1
## [163] memoise_1.1.0            dplyr_0.7.5
## [165] ape_5.1
```

Author contributions

FP, DR, KS, and EP performed the data analaysis and wrote the code portions of the workflow. 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).

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] 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`.

[5] 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.

[6] 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.

[7] 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`.

[8] 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`.

[9] 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`.