

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

Fanny Perraudeau<sup>1</sup>, Davide Rissò<sup>2</sup>, Kelly N. Street<sup>1</sup>, Elizabeth Purdom<sup>3</sup>, and Sandrine Dudoit<sup>4</sup>

<sup>1</sup>Graduate Group in Biostatistics, University of California, Berkeley

<sup>2</sup>Division of Biostatistics and Epidemiology, Department of Healthcare Policy and Research, Weill Cornell Medicine

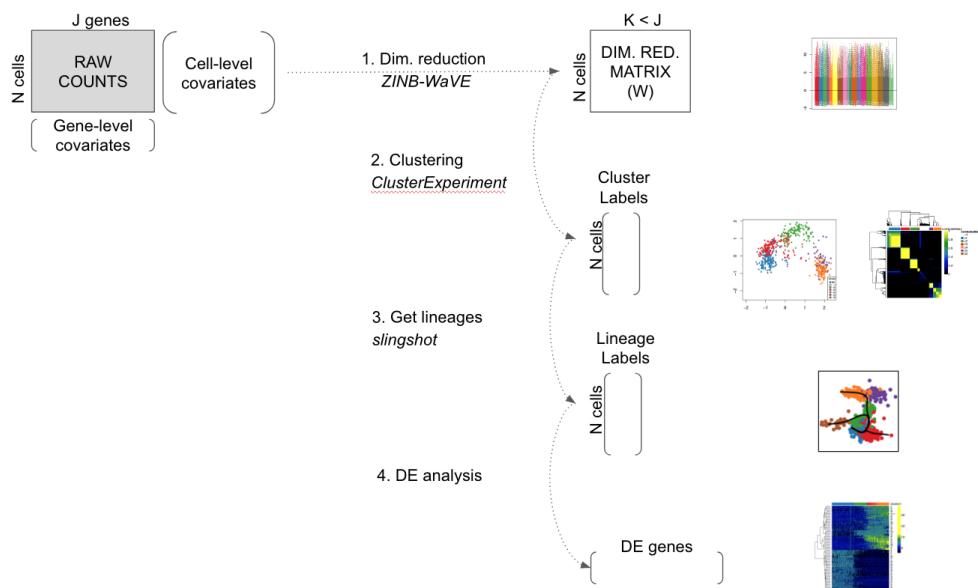
<sup>3</sup>Department of Statistics, University of California, Berkeley

<sup>4</sup>Graduate Group in Biostatistics, University of California, Berkeley; Division of Biostatistics and Department of Statistics, University of California, Berkeley

**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, single-cell genomics, RNA-seq, gene expression, zero inflation, normalization, dimensionality reduction, clustering, lineage inference, differential expression, workflow, R, Bioconductor



**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 differential expression (DE) analysis. 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 (A. T. Lun, McCarthy, and Marioni 2016) and the package `scater` (McCarthy et al. 2017) are such examples based on open-source R software packages from the Bioconductor Project (Huber et al. 2015). 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.

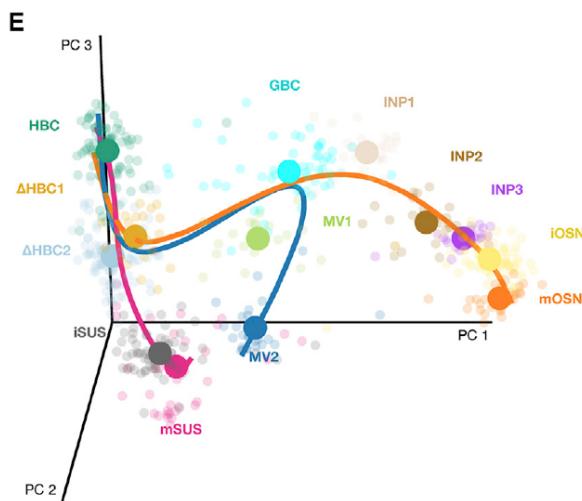
Here, we propose an integrated workflow for downstream analysis, with the following four main steps: (1) dimensionality reduction accounting for zero inflation and over-dispersion and adjusting for gene and cell-level covariates; (2) robust and stable cell clustering using resampling-based sequential ensemble clustering; (3) inference of cell lineages and ordering of the cells by developmental progression along lineages; and (4) DE analysis along lineages. Throughout the workflow, we use a unique `SummarizedExperiment` object to create an easy to use sequence of steps, where a different R package is used for each step.

## 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) (Fletcher et al. 2017). The olfactory epithelium contains mature olfactory sensory neurons that are continuously renewed in the epithelium via neurogenesis through the differentiation of globose basal cells (GBCs), 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 (HBCs), 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,



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

results were validated experimentally using *in vivo* lineage tracing. Details on data generation and statistical methods are available in (Fletcher et al. 2017), (Risso et al. 2017), and (K. Street et al. 2017). 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 mature olfactory sensory neurons (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  $\times$  cells matrix of raw counts publicly-available at <https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE95601>.

FP: ask Russell if we can use the Figure 2 here.

### 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).

```
load("../data/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 later corresponds to a marker gene for HBCs. Davide, is that correct?

```
# 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 an object of class `SummarizedExperiment` 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 (Fletcher et al. 2017). 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("../data/oeHBCdiff_clusterLabels.txt",
                           sep = "\t", stringsAsFactors = FALSE)
m <- match(colnames(E), clusterLabels[, 1])

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

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

se <- SummarizedExperiment(assays = list(counts = E),
                           colData = metadata)
se

## class: SummarizedExperiment
## 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

```

Using the Bioconductor R package `scone`, we remove low-quality cells that do not pass our quality control filter, that is ... See Figure ???. Davide, would you like to describe your choices for `scone` here?

```

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

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

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

```

Finally, to speed up computation, we retain only the 1,000 most variable genes. We recommend, however, not to be so stringent in an actual data analysis.

```

# Filtering to top 1000 most variable genes
vars <- rowVars(log1p(assay(se)))
names(vars) <- rownames(se)
vars <- sort(vars, decreasing = TRUE)
core <- se[names(vars)[1:1000],]

load("../data/oe_se_1000Var.rda")

```

## Dataset structure

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

```
core
```

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

Metadata for the cells are stored in the slot `colData` from the `SummarizedExperiment` object. Cells were processed in 18 different batches.

```
batch <- colData(core)$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 (Fletcher et al. 2017), cells were clustered into 14 different clusters, with 151 cells not assigned to any cluster (i.e., cluster label of -2).

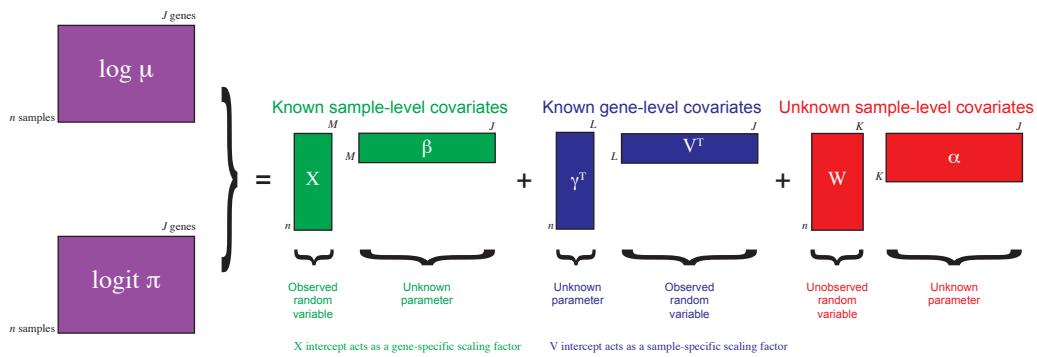
```
clus.labels <- colData(core)[, "clusterLabels"]
col_clus <- c("transparent", brewer.pal(12, "Set3"), brewer.pal(8, "Set2"))
col_clus <- col_clus[1:length(unique(clus.labels))]
names(col_clus) <- sort(unique(clus.labels))
table(clus.labels)
```

	## clus.labels	## -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.

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

	##	## 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	0	3	0
##	##	P01	7	0	2	4	3	15	0	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	



**Figure 3.** Schematic view of the ZINB-WaVE model. This figure was reproduced with kind permission from Risso et al. (2017).

### 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 (Dijk et al. 2017) (Risso et al. 2017) (Pierson and Yau 2015).

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 sample-level intercept, which serves as a global-scaling normalization factor. The user can also include both gene-level and sample-level covariates. The inclusion of observed and unobserved sample-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 3. For greater detail about the ZINB-WaVE model and estimation procedure, please refer to the original manuscript (Risso et al. 2017).

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.

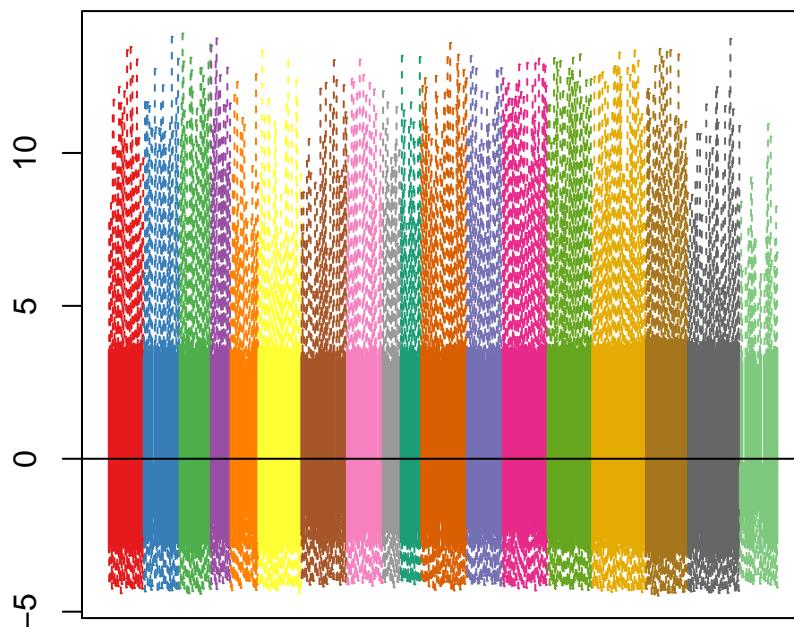
```
fn <- '../data/zinb_batch.rda'
if (!file.exists(fn)){
  print(system.time(se <- zinbDimRed(core, K = 50, X = 'Batch',
                                         residuals = TRUE,
                                         normalizedValues = TRUE)))
  save(se, file = fn)
} else{
  load(fn)
}
```

DR: the chunk above and the similar one for clusterExperiment are fine for now, but in the published workflow, we should just rely on the markdown cache system (the code above doesn't check for changes to the file or code – just that a version of the file exists). FP: Yes.

### Normalization

The function `zinbDimRed` returns normalized expression measures as deviance residuals from the fit of the ZINB-WaVE model, with user-supplied gene- and cell-level covariates. Such residuals can be used for visualization purposes (e.g., in heatmaps, boxplots). Note that in this case no low-dimensional matrix  $W$  is computed, as some of the biological signal of interest could be removed when computing residuals with respect to this matrix.

```
norm <- assays(se)$normalizedValues
if (sum(is.infinite(norm))>0){
  maxNorm = max(norm[!is.infinite(norm)])
  assays(se)$normalizedValues[is.infinite(norm)] <- maxNorm
  norm <- assays(se)$normalizedValues
}
norm[1:3,1:3]
```



**Figure 4.** Boxplots of normalized expression measures, color-coded by batch.

```
##          OEP01_N706_S501 OEP01_N701_S501 OEP01_N707_S507
## Cbr2        4.557371     4.375069    -4.142697
## Cyp2f2      4.321644     4.283266     4.090283
## Gstm1       4.796498     4.663366     4.416324
```

As expected, the normalized values no longer exhibit batch effects (Figure 4).

```
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')
abline(h=0)
```

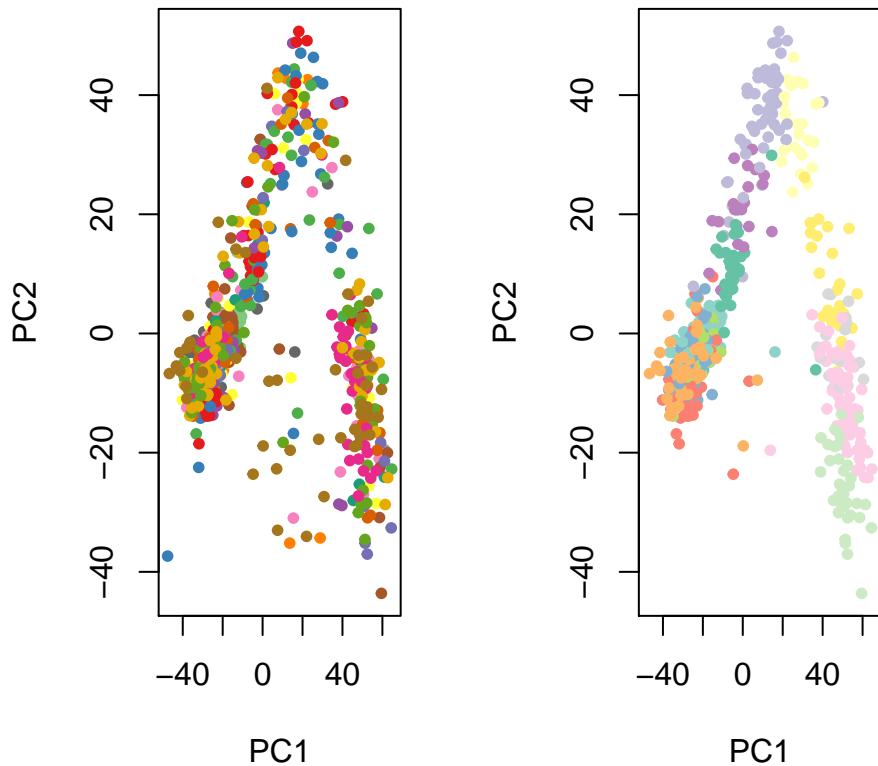
The principal component analysis (PCA) of the normalized values shows that, as expected, cells do not cluster by batch but by the original clusters (Figure 5). Overall, it seems that normalization was effective at removing batch effects.

```
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(clus.labels)], pch = 20, main = '')
```

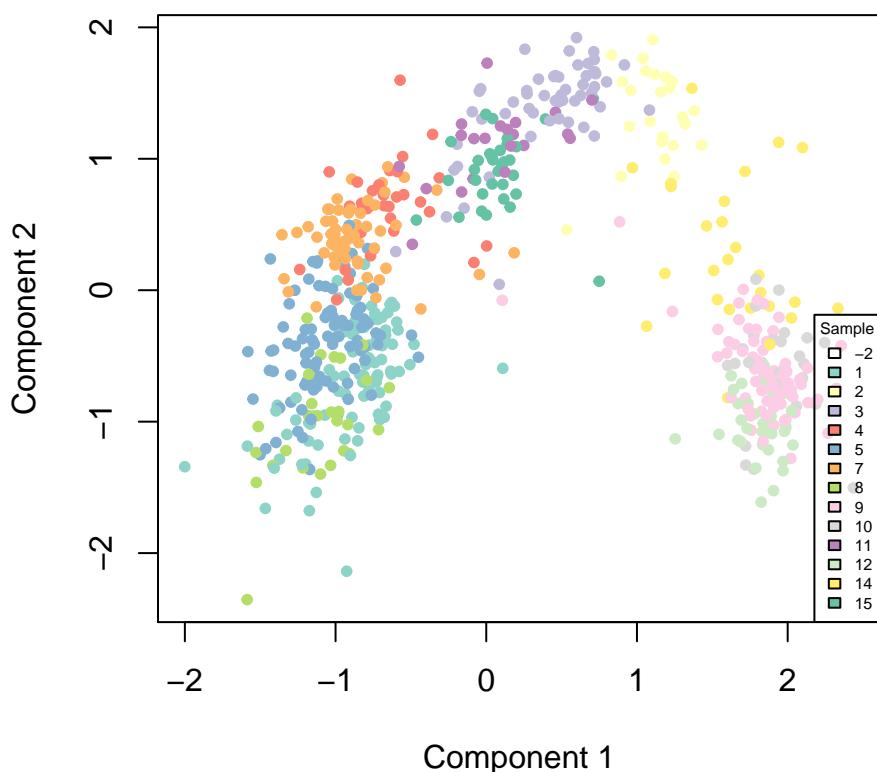
### Dimensionality reduction

The `zinbDimRed` 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 6).

```
W <- colData(se)[, grep('`W', colnames(colData(se)))]
W <- as.matrix(W)
d <- dist(W)
fit <- cmdscale(d, eig = TRUE, k = 2)
plot(fit$points, col = col_clus[as.character(clus.labels)], main = '',
      pch = 20, xlab = 'Component 1', ylab = 'Component 2')
legend(x = 'bottomright', legend = unique(names(col_clus)), cex = .5,
       fill = unique(col_clus), title = 'Sample')
```



**Figure 5.** PCA of normalized expression measures, where each point represents a cell. In the left panel, cells are color-coded by batch. In the right panel, cells are color-coded by original published clusters.



**Figure 6.** MDS of the low-dimensional matrix  $\mathbf{W}$ , where each point represents a cell and cells are color-coded by original published clusters.

### 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 Tseng and Wong (2005)(Tseng and Wong 2005). 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.

EP: Added the below paragraph

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 of the previous step, 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 (`dimReduce`) and the construction of the hierarchy between the clusters (`dendroReduce`). We also skip the option to merge our clusters based on the amount of differential gene expression between clusters since our input data is  $W$  and not the actual individual gene expression values.

```
fn <- 'data/RSEC_W_combineMinSize10.rda'
if (runClus & !file.exists(fn)){
  # Symbol for samples missing from original clustering
  seObj <- SummarizedExperiment(t(W), colData = colData(core))
  print(system.time(ceObj <- RSEC(seObj, k0s = 4:15, alphas = c(0.1),
                                    betas = 0.8, dimReduce="none",
                                    clusterFunction = "hierarchical01", minSizes=1,
                                    ncores = NCORES, isCount=FALSE,
                                    dendroReduce="none", dendroNDims=NA,
                                    subsampleArgs = list(resamp.num=100,
                                                         clusterFunction="kmeans",
                                                         clusterArgs=list(nstart=10)),
                                    verbose=TRUE,
                                    combineProportion = 0.7,
                                    mergeMethod = "none", random.seed=424242,
                                    combineMinSize = 10)))
  save(ceObj, file = fn)
} else{
  load(fn)
}
```

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.

```
plotClusters(ceObj, colPalette = c(bigPalette, rainbow(199)))
```

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).

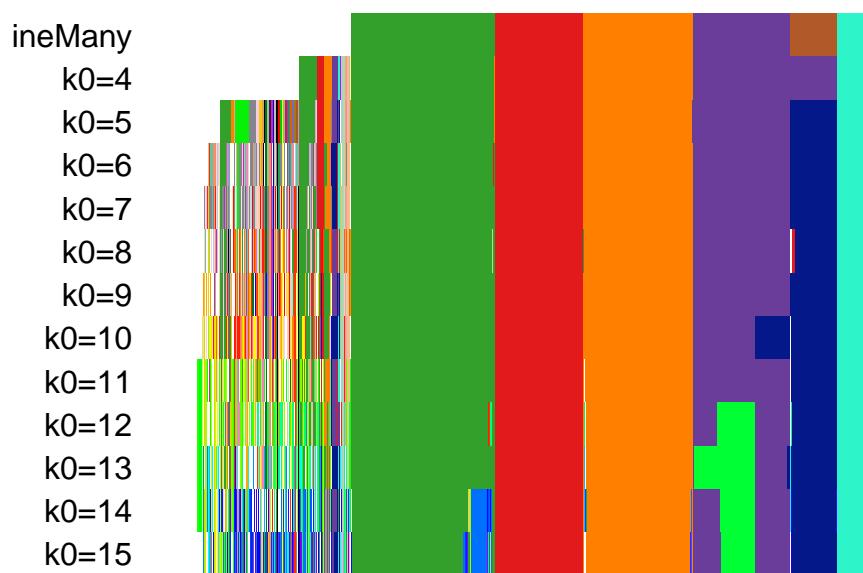
```
plotCoClustering(ceObj)
```

```
## Warning in .makeColors(clusters, colors = bigPalette): too many clusters to
## have unique color assignments
```

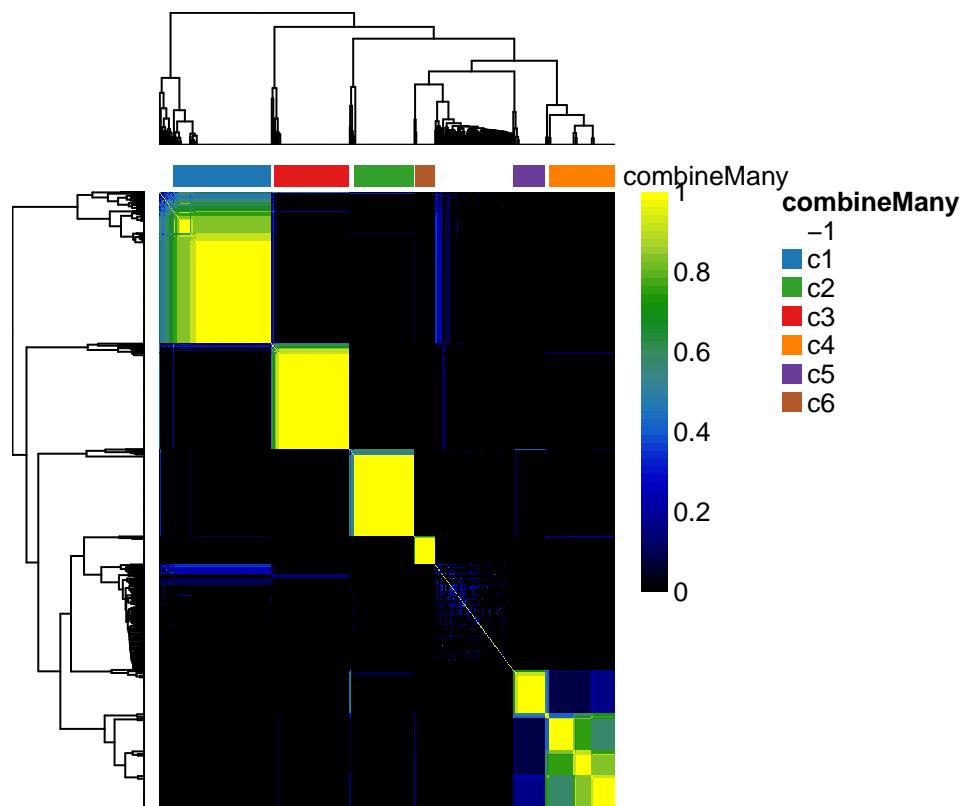
The distribution of cells across the consensus clusters is as follows:

```
table(primaryClusterNamed(ceObj))
```

```
##
##  -1  c1  c2  c3  c4  c5  c6
## 172 161  98 123 108  52  33
```



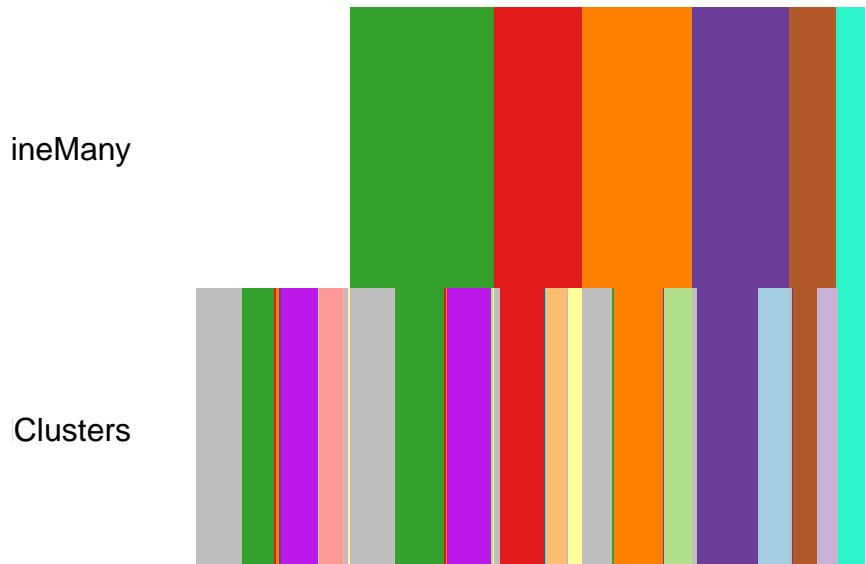
**Figure 7.** Clusters found using the function RSEC from the clusterExperiment package.



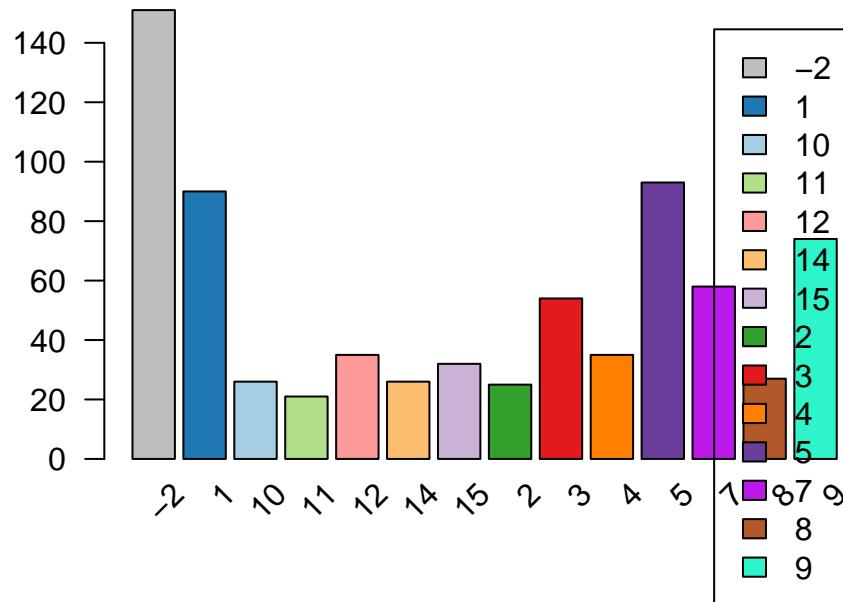
**Figure 8.** Heatmap of co-clustering matrix.

EP: the label `clusterLabels` for the original labels might do better to be changed to `publishedClusters` so that it is more informative. Also, we don't do a direct comparison of the `publishedClusters` to the found clusters – is that relevant? If so, it is probably better to add the published clusters via `addClusters` so it is easier to treat them like clusters. This is perhaps useful for a workflow since this is a common task that `clusterExperiment` makes easy. Here is code that does this:

```
r #delete this chunk if not of interest #if keep, need to choose if do both figures
and add figure caption ceObj<-addClusters(ceObj,colData(ceObj)$clusterLabels,clusterLabel="published")
plotClusters(ceObj, whichClusters=c("mergeClusters","combineMany","publishedClusters"))
```



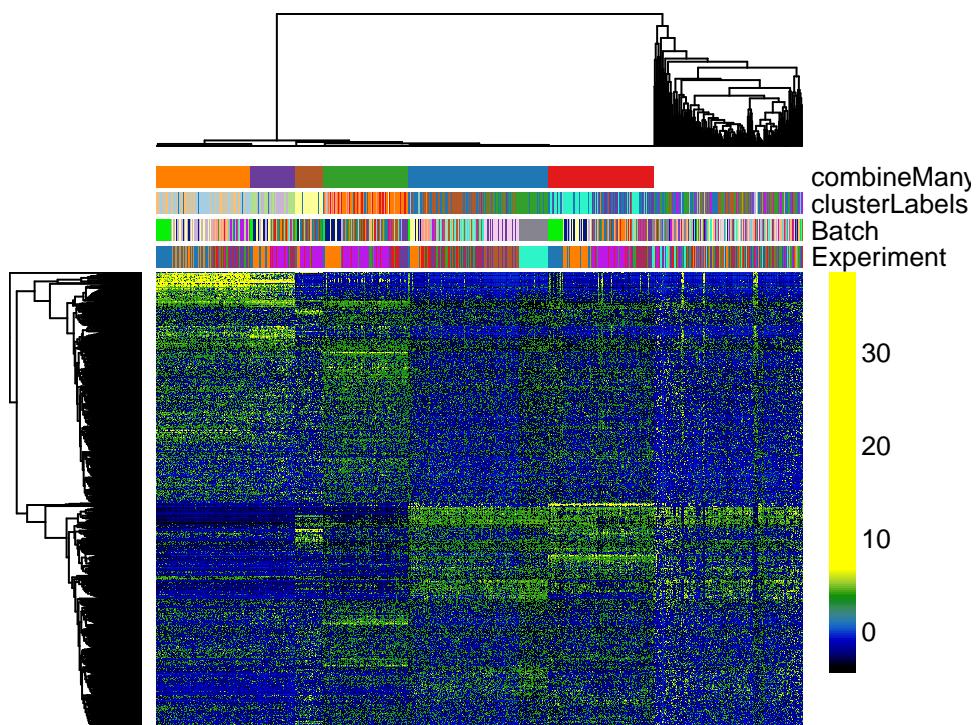
```
r plotBarplot(ceObj,whichClusters=c("mergeClusters","publishedClusters"))
```



EP: If we do the above code, then the `plotHeatmap` command below should be modified to have the `publishedClusters` under the `whichClusters` argument.

Figure 9 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 cell clusterings
colData(ceObj)$clusterLabels <- as.factor(colData(ceObj)$clusterLabels)
origClusterColors <- bigPalette[1:nlevels(colData(ceObj)$clusterLabels)]
experimentColors <- bigPalette[1:nlevels(colData(ceObj)$Experiment)]
batchColors <- bigPalette[1:nlevels(colData(ceObj)$Batch)]
metaColors <- list("Experiment" = experimentColors,
                    "Batch" = batchColors,
                    "clusterLabels" = origClusterColors)
```



**Figure 9.** 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.

```
plotHeatmap(ceObj, visualizeData = assays(se)$normalizedValues,
            whichClusters = "primary", clusterFeaturesData = "all",
            clusterSamplesData = "dendrogramValue", breaks = 0.99,
            sampleData = c("clusterLabels", "Batch", "Experiment"),
            clusterLegend = metaColors, annLegend = FALSE, main = '')
```

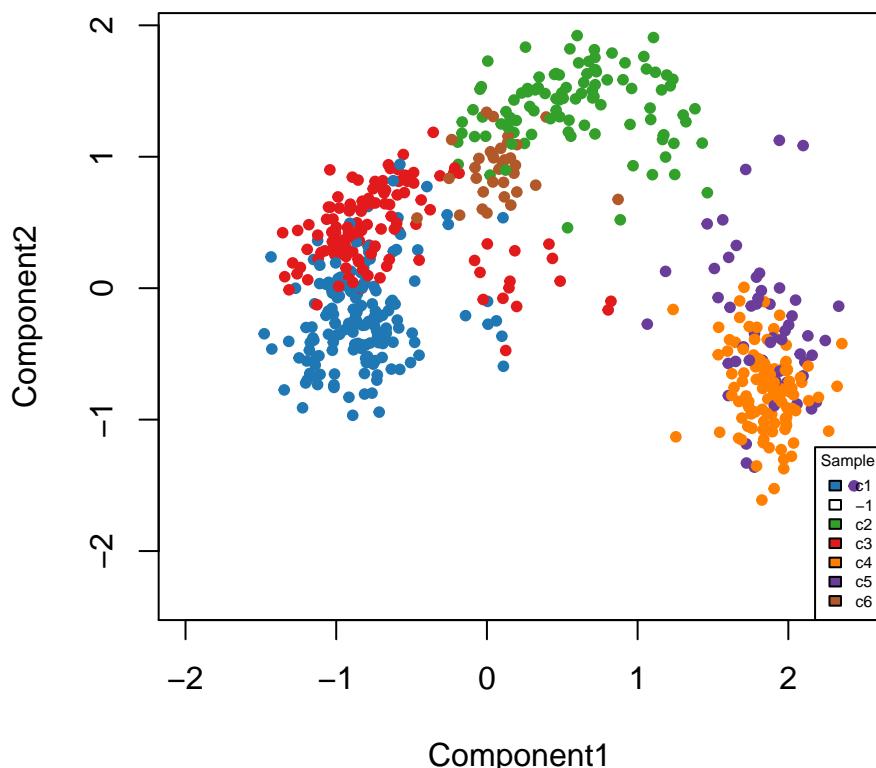
Finally, we can visualize the cells in a two-dimensional space using the MDS of the low-dimensional matrix  $W$  and coloring the cells according to their newly-found RSEC clusters (Figure 10); this is analogous to Figure 6 for the original published clusters.

```
palDF <- ceObj@clusterLegend[[1]]
pal <- palDF[, 'color']
names(pal) <- palDF[, 'name']
pal["-1"] = "transparent"
plot(fit$points, col = pal[primaryClusterNamed(ceObj)], main = '',
      pch = 20,
      xlab = 'Component1', ylab = 'Component2')
legend(x = 'bottomright', legend = names(pal), cex = .5,
       fill = pal, title = 'Sample')
```

#### 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 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 us to identify 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`. The first takes 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`.



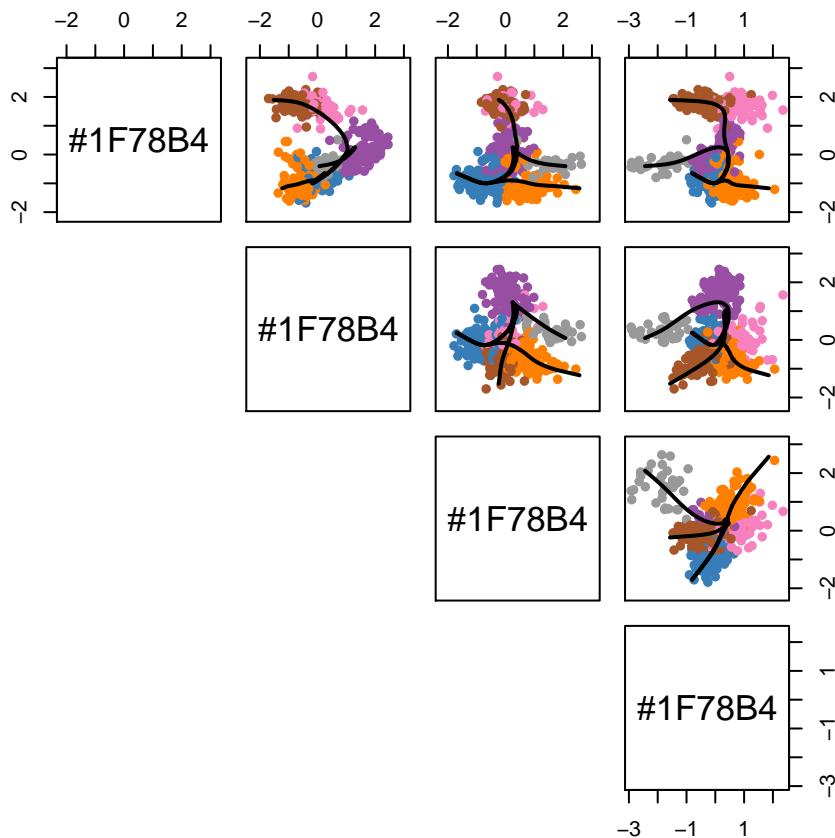
**Figure 10.** MDS of the low-dimensional matrix  $W$ , where each point represents a cell and cells are color-coded by RSEC clusters.

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

```
table(data.frame(original = clus.labels, ours = primaryClusterNamed(ce0bj)))
```

```
##          ours
## original -1 c1 c2 c3 c4 c5 c6
##      -2 51 50  6 33  5  3  3
##       1 41 49  0  0  0  0  0
##       2  1  0 24  0  0  0  0
##       3  2  2 49  1  0  0  0
##       4  1  2  0 32  0  0  0
##       5 36 55  0  2  0  0  0
##       7  3  1  0 54  0  0  0
##       8 27  0  0  0  0  0  0
##       9  2  0  1  1 68  2  0
##      10  0  0  0  0  0 26  0
##      11  3  2 16  0  0  0  0
##      12  0  0  0  0 35  0  0
##      14  4  0  1  0  0 21  0
##      15  1  0  1  0  0  0 30
```

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	mOSN	orange	original 9, 12
c5	Immature Neuron	purple	original 10, 14
c6	MV	brown	original 15



**Figure 11.** Cells color-coded by cluster in a 4-dimensional space with smooth curves representing each inferred lineage.

To infer lineages and pseudotimes, we use the package `slingshot`, with input 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. This implementation recovers the lineages found in the published work (Figures 11 and 12).

```
our_cl <- primaryClusterNamed(ceObj)
cl = our_cl[our_cl != "-1"]
pal = pal[names(pal) != '-1']
X <- W[our_cl != "-1", ]
mds <- cmdscale(dist(X), eig = TRUE, k = 4)
X <- mds$points

lineages <- getLineages(X, clusterLabels = cl, start.clus = "c1")
lineages <- getCurves(lineages)
```

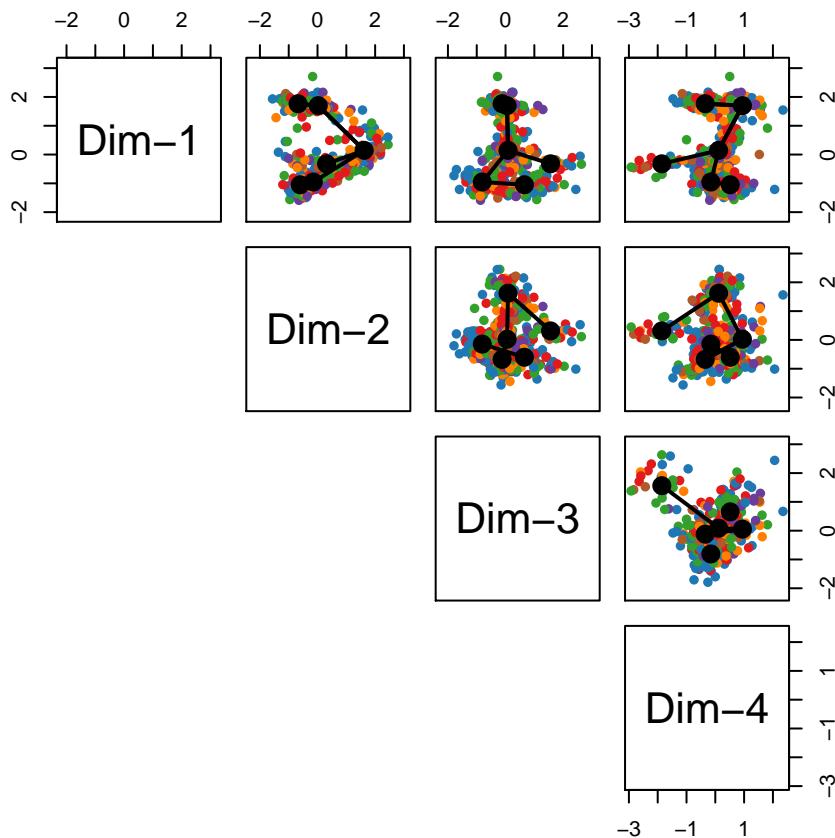
[SD: Mention reason for having these two figures, otherwise drop second figure. If keep second figure: Move it first, match notation for dimensions (along diagonal) in the two figures, and provide color legend for second figure.]

```
pairs(lineages, type='curves', pal[pri
```

```
pairs(lineages, type='lineages', col = pal[pri
```

```
lineages
```

```
## class: SlingshotDataSet
##
## Samples Dimensions
##      575          4
```



**Figure 12.** Cells color-coded by cluster in a 4-dimensional space with connecting lines between cluster centers representing the inferred structure.

```
##  
## lineages: 3  
## Lineage1: c1  c2  c5  c4  
## Lineage2: c1  c2  c6  
## Lineage3: c1  c3  
##  
## curves: 3  
## Curve1: Length: 8.0117    Samples: 374.34  
## Curve2: Length: 7.573     Samples: 283.32  
## Curve3: Length: 5.4656    Samples: 278.13
```

In the workflow, we recover a reasonable pseudotime 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.

```
lineages <- getLineages(X, clusterLabels = cl, start.clus = "c1",  
                           end.clus = c("c3", "c6"))  
lineagees <- getCurves(lineages)  
pairs(lineages, type='curves', pal[priorityClusterNamed(ceObj)])  
pairs(lineages, type='lineages', col = pal[priorityClusterNamed(ceObj)])  
  
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. We can see global trends in expression by subsetting our original dataset to the most variable genes [SD: How many genes?] and displaying their expression values in a heatmap, with cells ordered by pseudotime.

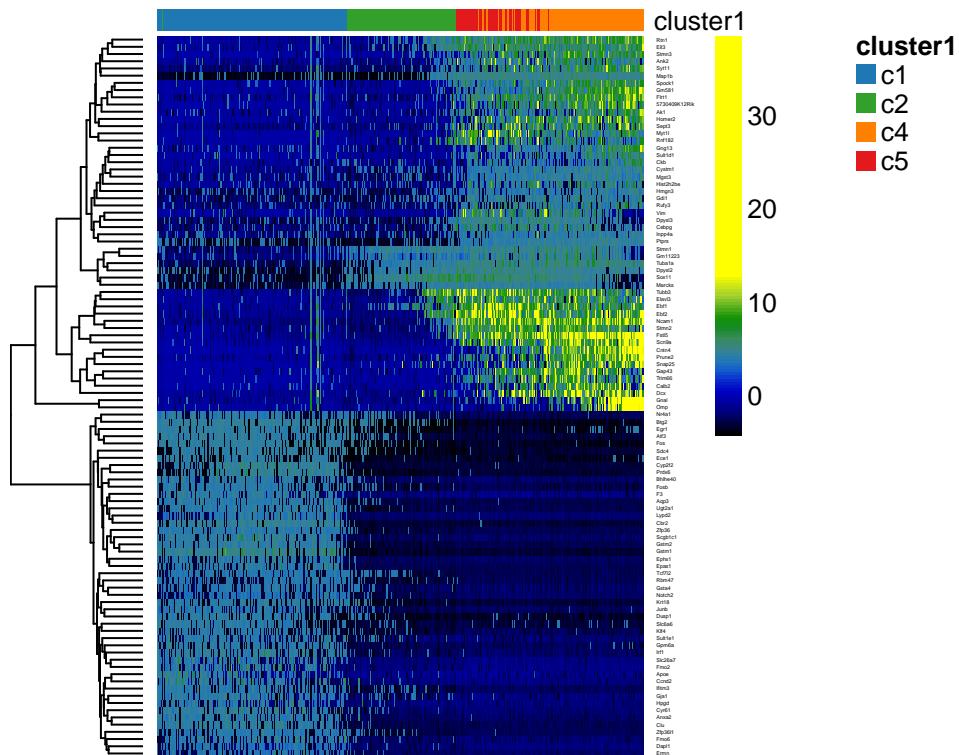
More formally, we use the robust local regression method loess to model in a flexible, non-linear manner the relationship between a gene's normalized expression measures and pseudotime. We then can test the null hypothesis of no change over time using the `gam` package. We implemented this approach for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

display the expression measures of the top 100 genes by p-value in the heatmap of Figure [SD: Add figure ref.].

```
t <- pseudotime(lineages)[,1]
y <- assays(se)$normalizedValues[, our_cl != "-1"]
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
})
```

[SD: Add figure caption.]

```
topgenes <- names(sort(gam.pval, decreasing = FALSE))[1:100]
heatdata <- y[rownames(se) %in% topgenes, order(t, na.last = NA)]
heatclus <- cl[order(t, na.last = NA)]
ce <- clusterExperiment(heatdata, heatclus, transformation = identity)
plotHeatmap(ce, clusterSamplesData = "orderSamplesValue", breaks = .99)
```



## 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. ZINB-WaVE can also be used for identifying genes that are differentially expressed between cells, 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 R class called `SingleCellExperiment` should be released soon. This new class would essentially be 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 this class is implemented, we would like to incorporate it to the workflow.

## Conclusion

This workflow provides a tutorial for the downstream analysis of scRNA-seq data in R. The workflow covers four main steps: (1) dimensionality reduction accounting for zero inflation and adjusting for gene and cell-level covariates; (2) robust and stable 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 within and between the 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 the technical aspect 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 four packages used in the workflow (scone, zinbwave, clusterExperiment, and slingshot) are Bioconductor packages and are available on github at respectively <https://github.com/YosefLab/scone>, <https://github.com/drisso/zinbwave>, <https://github.com/epurdom/clusterExperiment>, and <https://github.com/kstreet13/slingshot>. The source code for this package can be found at <https://github.com/fperraudeau/singlecellworkflow> under license XXX.

```
sessionInfo()
```

```
## R version 3.4.0 (2017-04-21)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS Sierra 10.12.5
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.4/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] gam_1.14-4           RColorBrewer_1.1-2
## [3] doParallel_1.0.10     iterators_1.0.8
## [5] foreach_1.4.3         slingshot_0.1.1
## [7] prcurve_1.1-12        zinbwave_0.99.4.2
## [9] BiocParallel_1.10.1    clusterExperiment_1.3.1
## [11] SummarizedExperiment_1.6.3 DelayedArray_0.2.7
## [13] matrixStats_0.52.2     Biobase_2.36.2
## [15] GenomicRanges_1.28.3   GenomeInfoDb_1.12.2
## [17] IRanges_2.10.2         S4Vectors_0.14.3
## [19] BiocGenerics_0.22.0    knitr_1.16
## [21] BiocStyle_2.4.0
##
## loaded via a namespace (and not attached):
## [1] colorspace_1.3-2       class_7.3-14
## [3] modeltools_0.2-21      mclust_5.3
## [5] rprojroot_1.2          XVector_0.16.0
## [7] gsl_1.9-10.3           flexmix_2.3-14
## [9] RSpecSpectra_0.12-0    mvtnorm_1.0-6
```

```

## [11] xml2_1.1.1           codetools_0.2-15
## [13] bold_0.4.0            robustbase_0.92-7
## [15] ade4_1.7-6             jsonlite_1.5
## [17] locfdr_1.1-8           phylobase_0.8.4
## [19] gridBase_0.4-7          cluster_2.0.6
## [21] kernlab_0.9-25         stabledist_0.7-1
## [23] shiny_1.0.3             copula_0.999-17
## [25] compiler_3.4.0          httr_1.2.1
## [27] backports_1.1.0         assertthat_0.2.0
## [29] Matrix_1.2-10           lazyeval_0.2.0
## [31] limma_3.32.2            htmltools_0.3.6
## [33] prettyunits_1.0.2         tools_3.4.0
## [35] igraph_1.0.1             bindrcpp_0.2
## [37] gtable_0.2.0              glue_1.1.1
## [39] taxize_0.8.4             GenomeInfoDbData_0.99.0
## [41] reshape2_1.4.2            dplyr_0.7.1
## [43] BiocWorkflowTools_1.2.0     Rcpp_0.12.11
## [45] softImpute_1.4             NMF_0.20.6
## [47] trimcluster_0.1-2          ape_4.1
## [49] nlme_3.1-131              fpc_2.1-10
## [51] stringr_1.2.0              mime_0.5
## [53] rngtools_1.2.4             XML_3.98-1.9
## [55] dendextend_1.5.2           DEoptimR_1.0-8
## [57] zlibbioc_1.22.0            MASS_7.3-47
## [59] scales_0.4.1              MAST_1.2.1
## [61] yaml_2.1.14                gridExtra_2.2.1
## [63] ggplot2_2.2.1              pkgmaker_0.22
## [65] reshape_0.8.6              stringi_1.1.5
## [67] pcaPP_1.9-72               rlang_0.1.1
## [69] pkgconfig_2.0.1            prabclus_2.2-6
## [71] bitops_1.0-6               rgl_0.98.1
## [73] rncl_0.8.2                 evaluate_0.10.1
## [75] lattice_0.20-35            bindr_0.1
## [77] htmlwidgets_0.8             plyr_1.8.4
## [79] magrittr_1.5                bookdown_0.4
## [81] R6_2.2.2                   ADGofTest_0.3
## [83] DBI_0.7                     whisker_0.3-2
## [85] abind_1.4-5                RCurl_1.95-4.8
## [87] nnet_7.3-12                 tibble_1.3.3
## [89] pspline_1.0-18              uuid_0.1-2
## [91] howmany_0.3-1               rmarkdown_1.6
## [93] viridis_0.4.0                progress_1.1.2
## [95] RNeXML_2.0.7                 grid_3.4.0
## [97] data.table_1.10.4            digest_0.6.12
## [99] diptest_0.75-7              xtable_1.8-2
## [101] httpuv_1.3.3               numDeriv_2016.8-1
## [103] tidyR_0.6.3                 glmnet_2.0-10
## [105] munsell_0.4.3               registry_0.3
## [107] viridisLite_0.2.0

```

### Author contributions

FP, DR, KNS, and EP performed the data analysis 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

[SD: Please check.]

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

## Acknowledgments

The authors are grateful to John Ngai (Department of Molecular and Cell Biology, UC Berkeley) and his group members Russell B. Fletcher and Diya Das for motivating the research presented in this workflow and for valuable feedback on applications to biological data.

## References

- Dijk, David van, Juozas Nainys, Roshan Sharma, Pooja Kathail, Ambrose J Carr, Kevin R Moon, Linas Mazutis, Guy Wolf, Smita Krishnaswamy, and Dana Pe'er. 2017. "MAGIC: A diffusion-based imputation method reveals gene-gene interactions in single-cell RNA-sequencing data." *BioRxiv*. doi:10.1101/111591.
- Fletcher, Russell B, Diya Das, Levi Gadye, Kelly N Street, Ariane Baudhuin, Allon Wagner, Michael B Cole, et al. 2017. "Deconstructing Olfactory Stem Cell Trajectories at Single-Cell Resolution." *Cell Stem Cell* 20 (6). Elsevier: 817–830.e8. doi:10.1016/j.stem.2017.04.003.
- Huber, Wolfgang, Vincent J Carey, Robert Gentleman, Simon Anders, Marc Carlson, Benilton S Carvalho, Hector Corrada Bravo, et al. 2015. "Orchestrating high-throughput genomic analysis with Bioconductor." *Nature Methods* 12 (2): 115–21. doi:10.1038/nmeth.3252.
- Lun, Aaron T.L., Davis J. McCarthy, and John C. Marioni. 2016. "A step-by-step workflow for low-level analysis of single-cell RNA-seq data with Bioconductor." *F1000Research* 5 (October): 2122. doi:10.12688/f1000research.9501.2.
- McCarthy, Davis J., Kieran R. Campbell, Aaron T. L. Lun, and Quin F. Wills. 2017. "Scater: pre-processing, quality control, normalization and visualization of single-cell RNA-seq data in R." *Bioinformatics*, January, btw777. doi:10.1093/bioinformatics/btw777.
- Pierson, Emma, and Christopher Yau. 2015. "ZIFA: Dimensionality reduction for zero-inflated single-cell gene expression analysis." *Genome Biology* 16 (1): 241. doi:10.1186/s13059-015-0805-z.
- Risso, Davide, Fanny Perraudeau, Svetlana Gribkova, Sandrine Dudoit, and Jean-Philippe Vert. 2017. "ZINB-WaVE: A general and flexible method for signal extraction from single-cell RNA-seq data." doi:10.1101/125112.
- Street, Kelly, Davide Risso, Russell B Fletcher, Diya Das, John Ngai, Nir Yosef, Elizabeth Purdom, and Sandrine Dudoit. 2017. "Slingshot: Cell lineage and pseudotime inference for single-cell transcriptomics." *BioRxiv*. <http://biorxiv.org/content/early/2017/04/19/128843.abstract>.
- Tseng, George C., and Wing H. Wong. 2005. "Tight Clustering: A Resampling-Based Approach for Identifying Stable and Tight Patterns in Data." *Biometrics* 61 (1): 10–16. doi:10.1111/j.0006-341X.2005.031032.x.