Analyzing single-cell RNA sequencing data from droplet-based protocols

Aaron T. L. Lun[1]

[1]Cancer Research UK Cambridge Institute, Li Ka Shing Centre, Robinson Way, Cambridge CB2 0RE, United Kingdom

2019-01-09

1 Overview

Droplet-based scRNA-seq protocols capture cells in droplets for massively multiplexed library prepation (Klein et al. 2015; Macosko et al. 2015). This greatly increases the throughput of scRNA-seq studies, allowing tens of thousands of individual cells to be profiled in a routine experiment. However, it (again) involves some differences from the previous workflows to reflect some unique aspects of droplet-based data.

Here, we describe a brief analysis of the peripheral blood mononuclear cell (PBMC) dataset from 10X Genomics (Zheng et al. 2017). The data are publicly available from the 10X Genomics website (https://support.10xgenomics.com/single-cell-gene-expression/datasets/2.1.0/pbmc4k), from which we download the raw gene/barcode count matrices, i.e., before cell calling from the *CellRanger* pipeline.

```
library(BiocFileCache)
bfc <- BiocFileCache("raw_data", ask = FALSE)
raw.path <- bfcrpath(bfc, file.path("http://cf.10xgenomics.c
om/samples",
    "cell-exp/2.1.0/pbmc4k/pbmc4k_raw_gene_bc_matrices.tar.g
z"))
untar(raw.path, exdir="pbmc4k")
```

2 Setting up the data

2.1 Reading in a sparse matrix

We load in the raw count matrix using the `read10xCounts()` function from the *DropletUtils (https://bioconductor.org/packages/3.8/DropletUtils)* package. This will create a `SingleCellExperiment` object where each column corresponds to a cell barcode.

```
library(DropletUtils)
fname <- "pbmc4k/raw_gene_bc_matrices/GRCh38"
sce <- read10xCounts(fname, col.names=TRUE)
sce
```

```
## class: SingleCellExperiment
## dim: 33694 737280
## metadata(0):
## assays(1): counts
## rownames(33694): ENSG00000243485 ENSG00000237613 ... ENSG
00000277475
##   ENSG00000268674
## rowData names(2): ID Symbol
## colnames(737280): AAACCTGAGAAACCAT-1 AAACCTGAGAAACCGC-1
...
##   TTTGTCATCTTTAGTC-1 TTTGTCATCTTTCCTC-1
## colData names(2): Sample Barcode
## reducedDimNames(0):
## spikeNames(0):
```

Here, each count represents the number of unique molecular identifiers (UMIs) assigned to a gene for a cell barcode. Note that the counts are loaded as a sparse matrix object - specifically, a `dgCMatrix` instance from the *Matrix (https://CRAN.R-project.org/package=Matrix)* package. This avoids allocating memory to hold zero counts, which is highly memory-efficient for low-coverage scRNA-seq data.

```
class(counts(sce))
```

```
## [1] "dgCMatrix"
## attr(,"package")
## [1] "Matrix"
```

2.2 Annotating the rows

We relabel the rows with the gene symbols for easier reading. This is done using the `uniquifyFeatureNames()` function, which ensures uniqueness in the case of duplicated or missing symbols.

```
library(scater)
rownames(sce) <- uniquifyFeatureNames(rowData(sce)$ID, rowDa
ta(sce)$Symbol)
head(rownames(sce))
```

```
## [1] "RP11-34P13.3"  "FAM138A"       "OR4F5"         "RP11
-34P13.7"
## [5] "RP11-34P13.8"  "RP11-34P13.14"
```

We also identify the chromosomal location for each gene. The mitochondrial location is particularly useful for later quality control.

```
library(EnsDb.Hsapiens.v86)
location <- mapIds(EnsDb.Hsapiens.v86, keys=rowData(sce)$ID,
    column="SEQNAME", keytype="GENEID")
rowData(sce)$CHR <- location
summary(location=="MT")
```

```
##    Mode   FALSE    TRUE    NA's
## logical   33537      13     144
```

3 Calling cells from empty droplets

3.1 Testing for deviations from ambient expression

An interesting aspect of droplet-based data is that we have no prior knowledge about which droplets (i.e., cell barcodes) actually contain cells, and which are empty. Thus, we need to call cells from empty droplets based on the observed expression profiles. This is not entirely straightforward as empty droplets can contain ambient (i.e., extracellular) RNA that can be captured and sequenced. The distribution of total counts exhibits a sharp transition between barcodes with large and small total counts (Figure 1), probably corresponding to cell-containing and empty droplets respectively.

```
bcrank <- barcodeRanks(counts(sce))

# Only showing unique points for plotting speed.
uniq <- !duplicated(bcrank$rank)
plot(bcrank$rank[uniq], bcrank$total[uniq], log="xy",
    xlab="Rank", ylab="Total UMI count", cex.lab=1.2)

abline(h=bcrank$inflection, col="darkgreen", lty=2)
abline(h=bcrank$knee, col="dodgerblue", lty=2)

legend("bottomleft", legend=c("Inflection", "Knee"),
    col=c("darkgreen", "dodgerblue"), lty=2, cex=1.2)
```

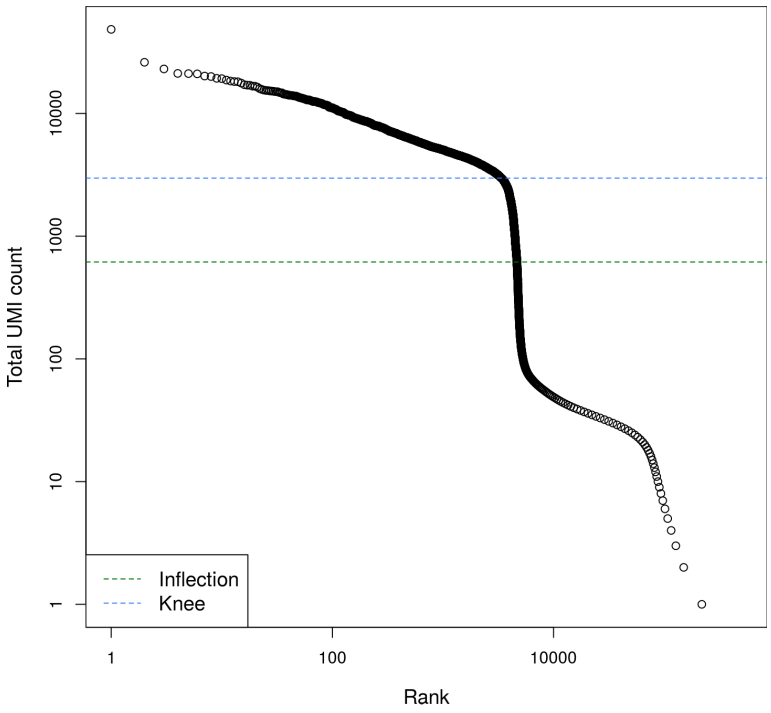

Figure 1: Total UMI count for each barcode in the PBMC dataset, plotted against its rank (in decreasing order of total counts). The inferred locations of the inflection and knee points are also shown.

We use the `emptyDrops()` function to test whether the expression profile for each cell barcode is significantly different from the ambient RNA pool (Lun et al. 2018). Any significant deviation indicates that the barcode corresponds to a cell-containing droplet. We call cells at a false discovery rate (FDR) of 1%, meaning that no more than 1% of our called barcodes should be empty droplets on average.

```
set.seed(100)
e.out <- emptyDrops(counts(sce))
sum(e.out$FDR <= 0.01, na.rm=TRUE)
```

```
## [1] 4358
```

We then subset our `SingleCellExperiment` object to retain only

the detected cells.

```
# using which() to automatically remove NAs.
sce <- sce[,which(e.out$FDR <= 0.01)]
```

Comments from Aaron:

- `emptyDrops()` computes Monte Carlo *p*-values based on a Dirichlet-multinomial model of sampling molecules into droplets. These *p*-values are stochastic so it is important to set the random seed to obtain reproducible results.
- The function assumes that cell barcodes with total UMI counts below a certain threshold (`lower=100` by default) correspond to empty droplets, and uses them to estimate the ambient expression profile. By definition, these barcodes cannot be cell-containing droplets and are excluded from the hypothesis testing, hence the `NA`s in the output.
- Users wanting to use the cell calling algorithm from the *CellRanger* pipeline can call `defaultDrops()` instead. This tends to be quite conservative as it often discards genuine cells with low RNA content (and thus low total counts). It also requires an estimate of the expected number of cells in the experiment.

3.2 Examining cell-calling diagnostics

The number of Monte Carlo iterations (specified by the `niters` argument in `emptyDrops()`) determines the lower bound for the _p_values (Phipson and Smyth 2010). The `Limited` field in the output indicates whether or not the computed *p*-value for a particular barcode is bounded by the number of iterations. If any non-significant barcodes are `TRUE` for `Limited`, we may need to increase the number of iterations. A larger number of iterations will often result in a lower *p*-value for these barcodes, which may allow them to be detected after correcting for multiple testing.

```
table(Sig=e.out$FDR <= 0.01, Limited=e.out$Limited)
```

```
##        Limited
## Sig     FALSE TRUE
##   FALSE   931    0
##   TRUE   1745 2613
```

As mentioned above, `emptyDrops()` assumes that barcodes with low total UMI counts are empty droplets. Thus, the null hypothesis should be true for all of these barcodes. We can check whether the hypothesis test holds its size by examining the distribution of *p*-values for low-total barcodes. Ideally, the distribution should be close to uniform.

```
full.data <- read10xCounts(fname, col.names=TRUE)
set.seed(100)
limit <- 100
all.out <- emptyDrops(counts(full.data), lower=limit, test.a
mbient=TRUE)
hist(all.out$PValue[all.out$Total <= limit & all.out$Total >
0],
    xlab="P-value", main="", col="grey80")
```

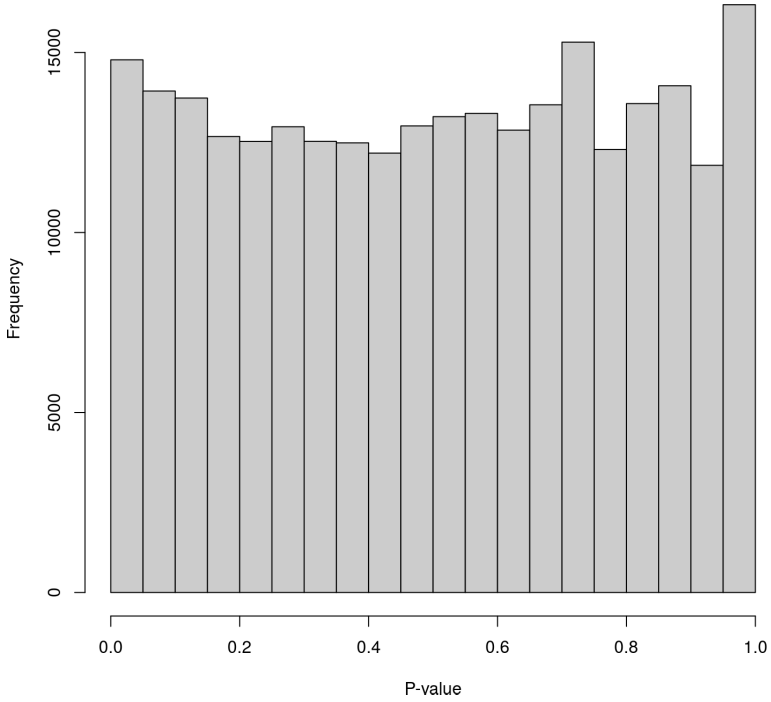

Figure 2: Distribution of p-values for the assumed empty droplets.

Large peaks near zero indicate that barcodes with total counts below `lower` are not all ambient in origin. This can be resolved by decreasing `lower` further to exclude barcodes corresponding to droplets with very small cells.

4 Quality control on the cells

The previous step only distinguishes cells from empty droplets, but makes no statement about the quality of the cells. It is entirely possible for droplets to contain damaged or dying cells, which need to be removed prior to downstream analysis. We compute some QC metrics using `calculateQCMetrics()` (McCarthy et al. 2017) and examine their distributions in Figure 3.

```
sce <- calculateQCMetrics(sce, feature_controls=list(Mito=wh
ich(location=="MT")))
par(mfrow=c(1,3))
hist(sce$log10_total_counts, breaks=20, col="grey80",
    xlab="Log-total UMI count")
hist(sce$log10_total_features_by_counts, breaks=20, col="gre
y80",
    xlab="Log-total number of expressed features")
hist(sce$pct_counts_Mito, breaks=20, col="grey80",
    xlab="Proportion of reads in mitochondrial genes")
```

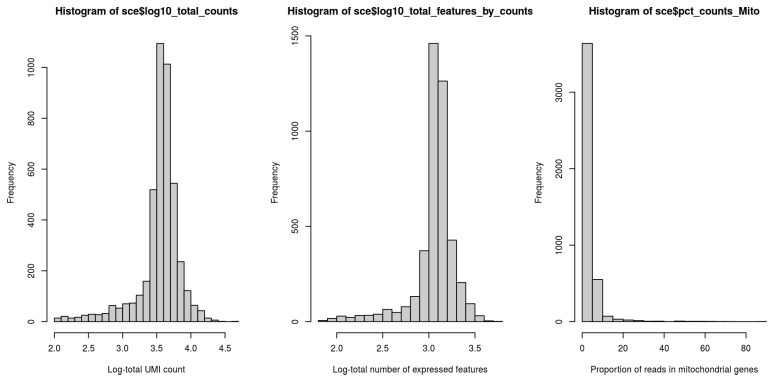

Figure 3: Histograms of QC metric distributions in the PBMC dataset.

Ideally, we would remove cells with low library sizes or total number of expressed features as described previously (https://bioconductor.org/packages/3.8/simpleSingleCell/vignettes/work-1-reads.html#quality-control-on-the-cells). However, this would likely remove cell types with low RNA content, especially in a heterogeneous PBMC population with many different cell types. Thus, we use a more relaxed strategy and only remove cells with large mitochondrial proportions, using it as a proxy for cell damage. (Keep in mind that droplet-based datasets usually do not have spike-in RNA.)

```
high.mito <- isOutlier(sce$pct_counts_Mito, nmads=3, type="h
igher")
sce <- sce[,!high.mito]
summary(high.mito)
```

```
##    Mode   FALSE    TRUE
## logical    4037     321
```

Comments from Aaron:

- The above justification for using a more relaxed filter is largely retrospective. In practice, we may not know *a priori* the degree of population heterogeneity and whether it manifests in the QC metrics. We recommend performing the analysis first with a stringent QC filter, and then relaxing it based on further diagnostics (see here (https://bioconductor.org/packages/3.8/simpleSingleCell

/vignettes/xtra-1-qc.html#checking-for-discarded-cell-types) for an example).

5 Examining gene expression

The average expression of each gene is much lower here compared to the previous datasets (Figure 4). This is due to the reduced coverage per cell when thousands of cells are multiplexed together for sequencing.

```
ave <- calcAverage(sce)
rowData(sce)$AveCount <- ave
hist(log10(ave), col="grey80")
```

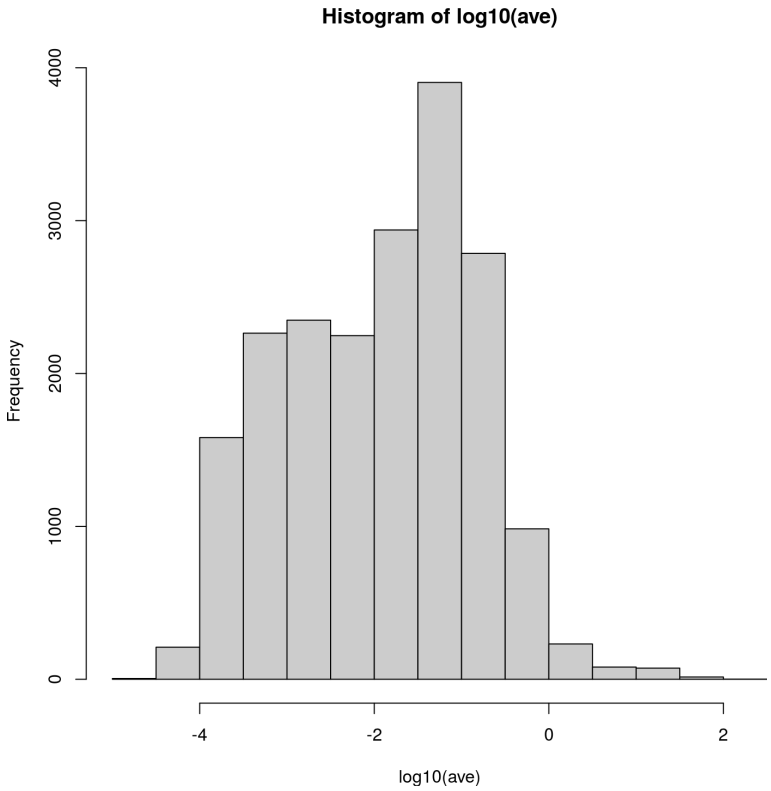

Figure 4: Histogram of the $\log_{10}$-average counts for each gene in the PBMC dataset.

The set of most highly expressed genes is dominated by ribosomal protein and mitochondrial genes (Figure 5), as expected.

```
plotHighestExprs(sce)
```

Figure 5: Percentage of total counts assigned to the top 50 most highly-abundant features in the PBMC dataset. For each feature, each bar represents the percentage assigned to that feature for a single cell, while the circle represents the average across all cells. Bars are coloured by the total number of expressed features in each cell.

6 Normalizing for cell-specific biases

We perform some pre-clustering to break up obvious clusters, as described previously (https://bioconductor.org/packages/3.8/simpleSingleCell/vignettes/work-2-umis.html#normalization-of-cell-specific-biases). Recall that we need to set the seed when using `method="igraph"` .

```
library(scran)
set.seed(1000)
clusters <- quickCluster(sce, method="igraph", min.mean=0.1,
    irlba.args=list(maxit=1000)) # for convergence.
table(clusters)
```

```
## clusters
##    1    2    3    4    5
##  937  269 1173 1109  549
```

We apply the deconvolution method to compute size factors for all cells (Lun, Bach, and Marioni 2016). The specification of `cluster=` ensures that we do not pool cells that are very different.

```
sce <- computeSumFactors(sce, min.mean=0.1, cluster=cluster
s)
summary(sizeFactors(sce))
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Ma
x.
##  0.005117  0.722562  0.884547  1.000000  1.105609 12.7117
94
```

The size factors are well correlated against the library sizes (Figure 6), indicating that capture efficiency and sequencing depth are the major biases.

```
plot(sce$total_counts, sizeFactors(sce), log="xy")
```

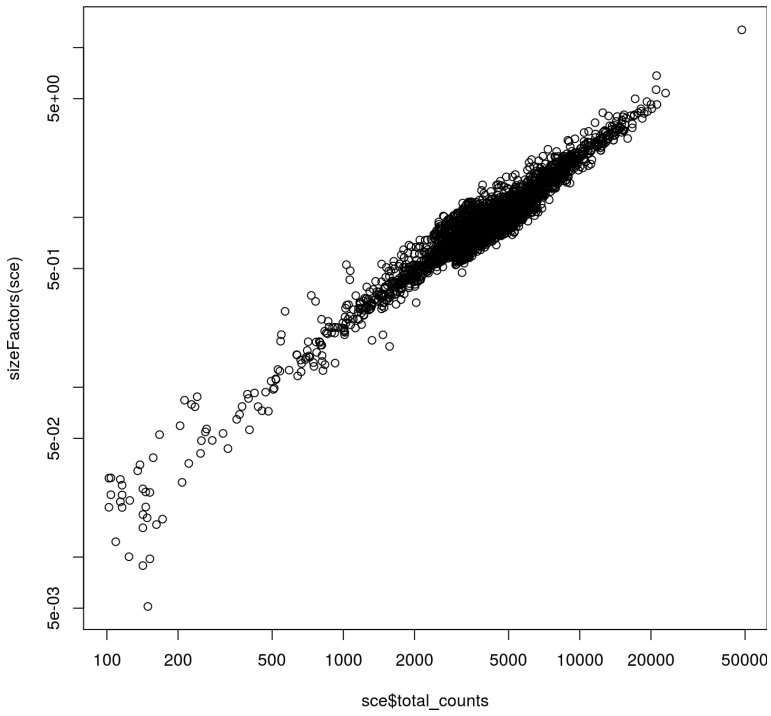

Figure 6: Size factors for all cells in the PBMC dataset, plotted against the library

size.

Finally, we compute normalized log-expression values. There is no need to call `computeSpikeFactors()` here, as there are no spike-in transcripts available.

```
sce <- normalize(sce)
```

Comments from Aaron:

- Larger droplet-based datasets will often be generated in separate batches or runs. In such cases, we can set `block=` in `quickCluster()` to cluster cells within each batch or run. This reduces computational work considerably without compromising performance, provided that the clusters within each batch are sufficiently large (see comments here (https://bioconductor.org/packages/3.8/simpleSingleCell/vignettes/work-2-umis.html#6_normalization_of_cell-specific_biases)) for a discussion of the considerations involved in pre-clustering for normalization).
- Even in the absence of any known batch structure, we can improve speed by setting an arbitrary factor, e.g., using `block=cut(seq_len(ncol(sce)), 10)` to split the cells into ten "batches" of roughly equal size. Recall that we are not interpreting the clusters themselves, so it is not a problem to have multiple redundant cluster labels. Again, this assumes that each cluster is large enough to support deconvolution.
- On a similar note, both `quickCluster()` and `computeSumFactors()` can process blocks or clusters in parallel. This is achieved using the *BiocParallel (https://bioconductor.org/packages/3.8/BiocParallel)* framework, which accommodates a range of parallelization strategies. In this manner, size factors for large datasets can be computed in a scalable manner.

7 Modelling the mean-variance trend

The lack of spike-in transcripts complicates the modelling of the technical noise. One option is to assume that most genes do not exhibit strong biological variation, and to fit a trend to the variances of endogenous genes (see here (https://bioconductor.org/packages/3.8/simpleSingleCell/vignettes/xtra-3-var.html#32_when_spike-ins_are_unavailable) for details). However, this assumption is generally unreasonable for a heterogeneous population. Instead, we assume that the technical noise is Poisson and create a fitted trend on that basis using the `makeTechTrend()` function.

```
new.trend <- makeTechTrend(x=sce)
```

We estimate the variances for all genes and compare the trend fits in Figure 7. The Poisson-based trend serves as a lower bound for the variances of the endogenous genes. This results in non-zero biological components for most genes, which is consistent with other UMI-based data sets (see the corresponding analysis (https://bioconductor.org/packages/3.8/simpleSingleCell/vignettes/work-2-umis.html#7_modelling_and_removing_technical_noise) of the Zeisel et al. (2015) data set).

```
fit <- trendVar(sce, use.spikes=FALSE, loess.args=list(span=
0.05))
plot(fit$mean, fit$var, pch=16)
curve(fit$trend(x), col="dodgerblue", add=TRUE)
curve(new.trend(x), col="red", add=TRUE)
```

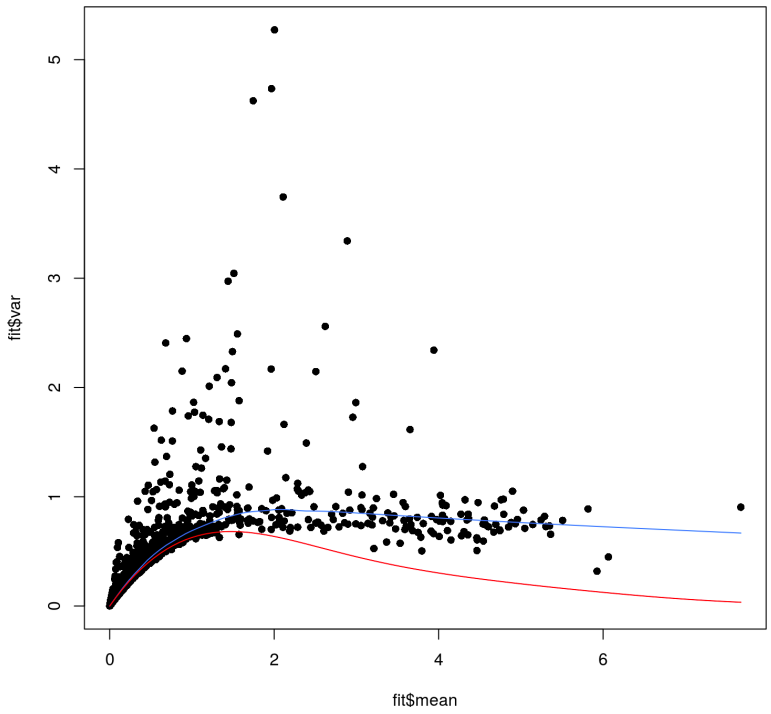

Figure 7: Variance of normalized log-expression values for each gene in the PBMC dataset, plotted against the mean log-expression. The blue line represents the mean-dependent trend fitted to the variances, while the red line represents the Poisson noise.

We decompose the variance for each gene using the Poisson-based trend, and examine the genes with the highest biological components.

```
fit0 <- fit
fit$trend <- new.trend
dec <- decomposeVar(fit=fit)
top.dec <- dec[order(dec$bio, decreasing=TRUE),]
head(top.dec)
```

```
## DataFrame with 6 rows and 6 columns
##                      mean            total              bi
o              tech
##                 <numeric>        <numeric>        <numeric
>         <numeric>
## LYZ     2.00433282867795 5.27238976970295 4.6361116933853
8 0.636278076317563
## S100A9  1.96814061490774 4.73501347757919 4.0933522680087
9 0.641661209570392
## S100A8   1.7450872893628 4.62384461299396 3.9552414733582
3 0.668603139635734
## HLA-DRA 2.10913349062558 3.74365888475255 3.1241987070064
3 0.619460177746116
## CD74    2.88808019297542 3.34102575092047 2.8696215546565
8 0.471404196263887
## CST3    1.51005991810233 3.04448800763578 2.3629366835770
4 0.681551324058745
##           p.value       FDR
##         <numeric> <numeric>
## LYZ             0         0
## S100A9          0         0
## S100A8          0         0
## HLA-DRA         0         0
## CD74            0         0
## CST3            0         0
```

We can plot the genes with the largest biological components, to verify that they are indeed highly variable (Figure 8).

```
plotExpression(sce, features=rownames(top.dec)[1:10])
```