Analyzing single-cell RNA-seq data containing UMI counts

Aaron T. L. Lun[1]***, Davis J. McCarthy***[2,3] ***and John C. Marioni***[1,2,4]

[1]Cancer Research UK Cambridge Institute, Li Ka Shing Centre, Robinson Way, Cambridge CB2 0RE, United Kingdom
[2]EMBL European Bioinformatics Institute, Wellcome Genome Campus, Hinxton, Cambridge CB10 1SD, United Kingdom
[3]St Vincent's Institute of Medical Research, 41 Victoria Parade, Fitzroy, Victoria 3065, Australia
[4]Wellcome Trust Sanger Institute, Wellcome Genome Campus, Hinxton, Cambridge CB10 1SA, United Kingdom

2019-01-09

1 Overview

In this workflow, we examine a heterogeneous dataset from a study of cell types in the mouse brain (Zeisel et al. 2015). This contains approximately 3000 cells of varying types such as oligodendrocytes, microglia and neurons. Individual cells were isolated using the Fluidigm C1 microfluidics system (Pollen et al. 2014) and library preparation was performed on each cell using a UMI-based protocol. After sequencing, expression was quantified by counting the number of UMIs mapped to each gene. Count data for all endogenous genes, mitochondrial genes and spike-in transcripts are available from http://linnarssonlab.org/cortex

(http://linnarssonlab.org/cortex).

```
library(BiocFileCache)
bfc <- BiocFileCache("raw_data", ask = FALSE)
base.url <- file.path("https://storage.googleapis.com",
    "linnarsson-lab-www-blobs/blobs/cortex")
mRNA.path <- bfcrpath(bfc, file.path(base.url,
    "expression_mRNA_17-Aug-2014.txt"))
mito.path <- bfcrpath(bfc, file.path(base.url,
    "expression_mito_17-Aug-2014.txt"))
spike.path <- bfcrpath(bfc, file.path(base.url,
    "expression_spikes_17-Aug-2014.txt"))
```

2 Setting up the data

The count data are distributed across several files, so some work is necessary to consolidate them into a single matrix. We define a simple utility function for loading data in from each file. (We stress that this function is only relevant to the current dataset, and should not be used for other datasets. This kind of effort is generally not required if all of the counts are in a single file and separated from the metadata.)

```
readFormat <- function(infile) {
    # First column is empty.
    metadata <- read.delim(infile, stringsAsFactors=FALSE, h
eader=FALSE, nrow=10)[,-1]
    rownames(metadata) <- metadata[,1]
    metadata <- metadata[,-1]
    metadata <- as.data.frame(t(metadata))

    # First column after row names is some useless filler.
    counts <- read.delim(infile, stringsAsFactors=FALSE,
        header=FALSE, row.names=1, skip=11)[,-1]
    counts <- as.matrix(counts)
    return(list(metadata=metadata, counts=counts))
}
```

Using this function, we read in the counts for the endogenous genes, ERCC spike-in transcripts and mitochondrial genes.

```
endo.data <- readFormat(mRNA.path)
spike.data <- readFormat(spike.path)
mito.data <- readFormat(mito.path)
```

We also need to rearrange the columns for the mitochondrial data, as the order is not consistent with the other files.

```
m <- match(endo.data$metadata$cell_id, mito.data$metadata$ce
ll_id)
mito.data$metadata <- mito.data$metadata[m,]
mito.data$counts <- mito.data$counts[,m]
```

In this particular dataset, some genes are represented by multiple rows corresponding to alternative genomic locations. We sum the counts for all rows corresponding to a single gene for ease of interpretation.

```
raw.names <- sub("_loc[0-9]+$", "", rownames(endo.data$count
s))
new.counts <- rowsum(endo.data$counts, group=raw.names, reor
der=FALSE)
endo.data$counts <- new.counts
```

The counts are then combined into a single matrix for constructing a `SingleCellExperiment` object. For convenience, metadata for all cells are stored in the same object for later access.

```
library(SingleCellExperiment)
all.counts <- rbind(endo.data$counts, mito.data$counts, spik
e.data$counts)
sce <- SingleCellExperiment(list(counts=all.counts), colData
=endo.data$metadata)
dim(sce)
```

```
## [1] 19896  3005
```

We add gene-based annotation identifying rows that correspond to each class of features. We also determine the Ensembl identifier for each row.

```
# Specifying the nature of each row.
nrows <- c(nrow(endo.data$counts), nrow(mito.data$counts), n
row(spike.data$counts))
is.spike <- rep(c(FALSE, FALSE, TRUE), nrows)
is.mito <- rep(c(FALSE, TRUE, FALSE), nrows)
isSpike(sce, "Spike") <- is.spike

# Adding Ensembl IDs.
library(org.Mm.eg.db)
ensembl <- mapIds(org.Mm.eg.db, keys=rownames(sce), keytype
="SYMBOL", column="ENSEMBL")
rowData(sce)$ENSEMBL <- ensembl

sce
```

```
## class: SingleCellExperiment
## dim: 19896 3005
## metadata(0):
## assays(1): counts
## rownames(19896): Tspan12 Tshz1 ... ERCC-00170 ERCC-00171
## rowData names(1): ENSEMBL
## colnames(3005): V3 V4 ... V3006 V3007
## colData names(10): tissue group # ... level1class level2c
lass
## reducedDimNames(0):
## spikeNames(1): Spike
```

3 Quality control on the cells

The original authors of the study have already removed low-quality cells prior to data publication. Nonetheless, we compute some quality control metrics with *scater (https://bioconductor.org/packages/3.8/scater)* (McCarthy et al. 2017) to check whether the remaining cells are satisfactory.

```
library(scater)
sce <- calculateQCMetrics(sce, feature_controls=list(Mt=is.m
ito))
```

We examine the distribution of the QC metrics across all cells (Figure 1). The library sizes here are at least one order of magnitude lower than observed in the 416B dataset. This is consistent with the use of UMI counts rather than read counts, as each transcript molecule can only produce one UMI count but can yield many reads after fragmentation. In addition, the spike-in proportions are more variable than observed in the 416B dataset. This may reflect a greater variability in the total amount of endogenous RNA per cell when many cell types are present.

```
par(mfrow=c(2,2), mar=c(5.1, 4.1, 0.1, 0.1))
hist(sce$total_counts/1e3, xlab="Library sizes (thousands)",
main="",
    breaks=20, col="grey80", ylab="Number of cells")
hist(sce$total_features_by_counts, xlab="Number of expressed
genes", main="",
    breaks=20, col="grey80", ylab="Number of cells")
hist(sce$pct_counts_Spike, xlab="ERCC proportion (%)",
    ylab="Number of cells", breaks=20, main="", col="grey8
0")
hist(sce$pct_counts_Mt, xlab="Mitochondrial proportion (%)",
    ylab="Number of cells", breaks=20, main="", col="grey8
0")
```

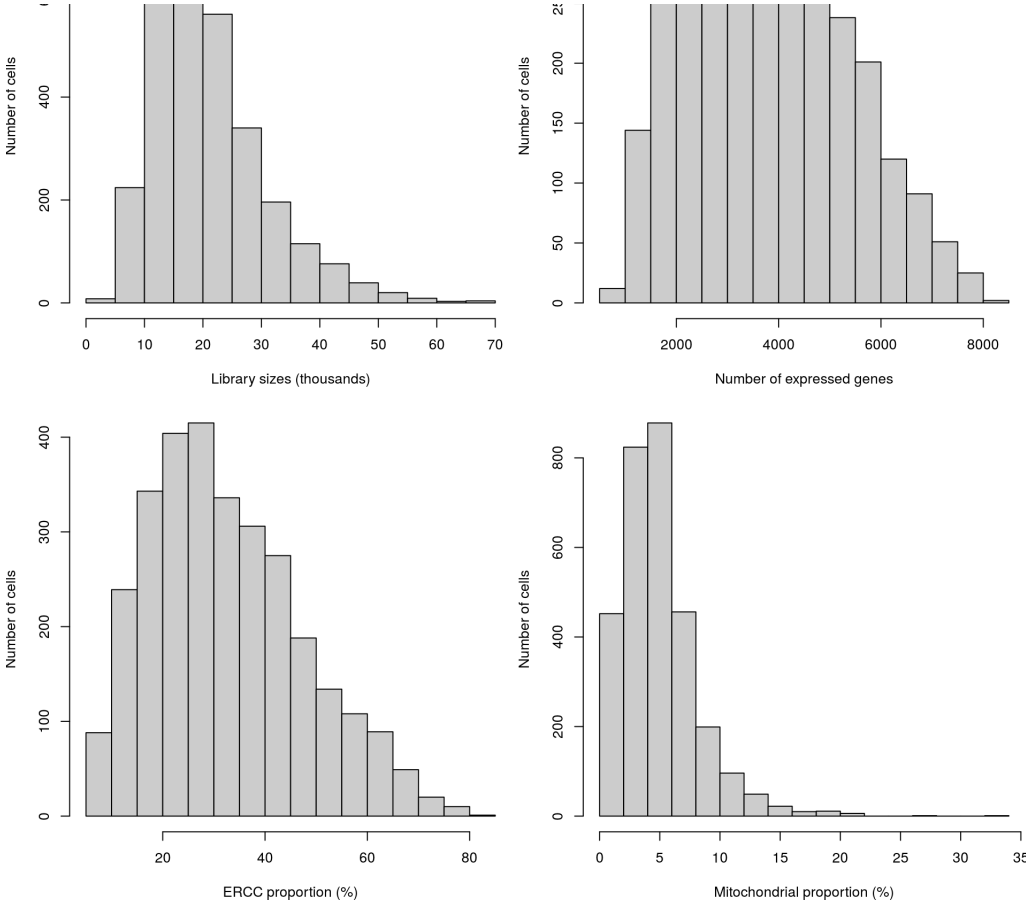


Figure 1: Histograms of QC metrics including the library sizes, number of expressed genes and proportion of UMIs assigned to spike-in transcripts or mitochondrial genes for all cells in the brain dataset.

We remove small outliers for the library size and the number of expressed features, and large outliers for the spike-in proportions. Again, the presence of spike-in transcripts means that we do not have to use the mitochondrial proportions.

```
libsize.drop <- isOutlier(sce$total_counts, nmads=3, type="l
ower", log=TRUE)
feature.drop <- isOutlier(sce$total_features_by_counts, nmad
s=3, type="lower", log=TRUE)
spike.drop <- isOutlier(sce$pct_counts_Spike, nmads=3, type
="higher")
```

Removal of low-quality cells is then performed by combining the filters for all of the metrics. The majority of cells are retained, which suggests that the original quality control procedures were generally adequate.

```
sce <- sce[,!(libsize.drop | feature.drop | spike.drop)]
data.frame(ByLibSize=sum(libsize.drop), ByFeature=sum(featur
e.drop),
    BySpike=sum(spike.drop), Remaining=ncol(sce))
```

```
##   ByLibSize ByFeature BySpike Remaining
## 1         8         3       8      2989
```

We could improve our cell filtering procedure further by setting `batch` in `isOutlier` to one or more known factors, e.g., mouse/plate of origin. As previously mentioned, this would avoid inflation of the MAD and improve power to remove low-quality cells. However, for simplicity, we will not do this as sufficient quality control has already been performed.

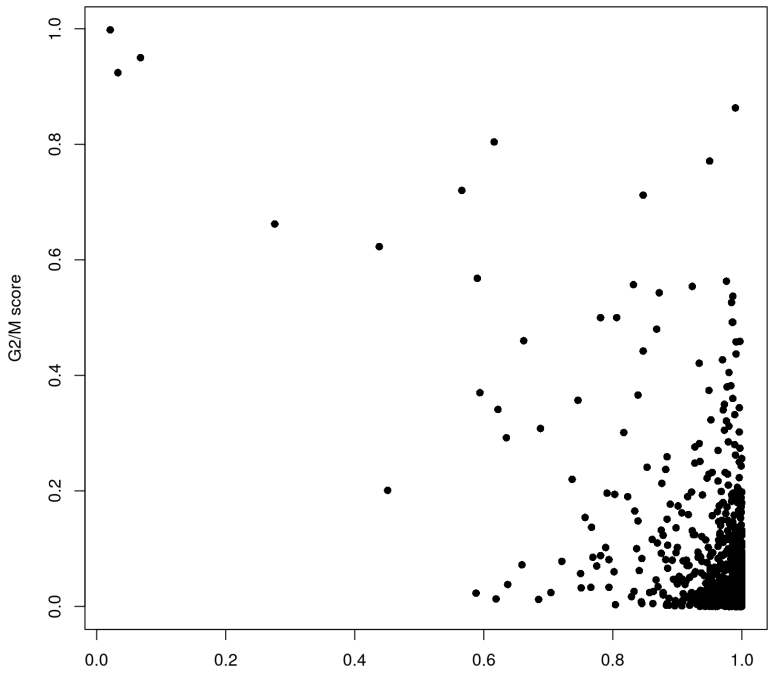
4 Cell cycle classification

Application of `cyclone` (Scialdone et al. 2015) to the brain dataset suggests that most of the cells are in G1 phase (Figure 2). This requires the use of the Ensembl identifiers to match up with the pre-defined classifier.

```
library(scran)
mm.pairs <- readRDS(system.file("exdata", "mouse_cycle_marke
rs.rds", package="scran"))
assignments <- cyclone(sce, mm.pairs, gene.names=rowData(sc
e)$ENSEMBL)
table(assignments$phase)
```

```
##
##   G1  G2M    S
## 2981    7    1
```

```
plot(assignments$score$G1, assignments$score$G2M, xlab="G1 s
core", ylab="G2/M score", pch=16)
```



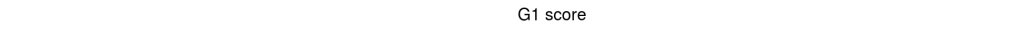


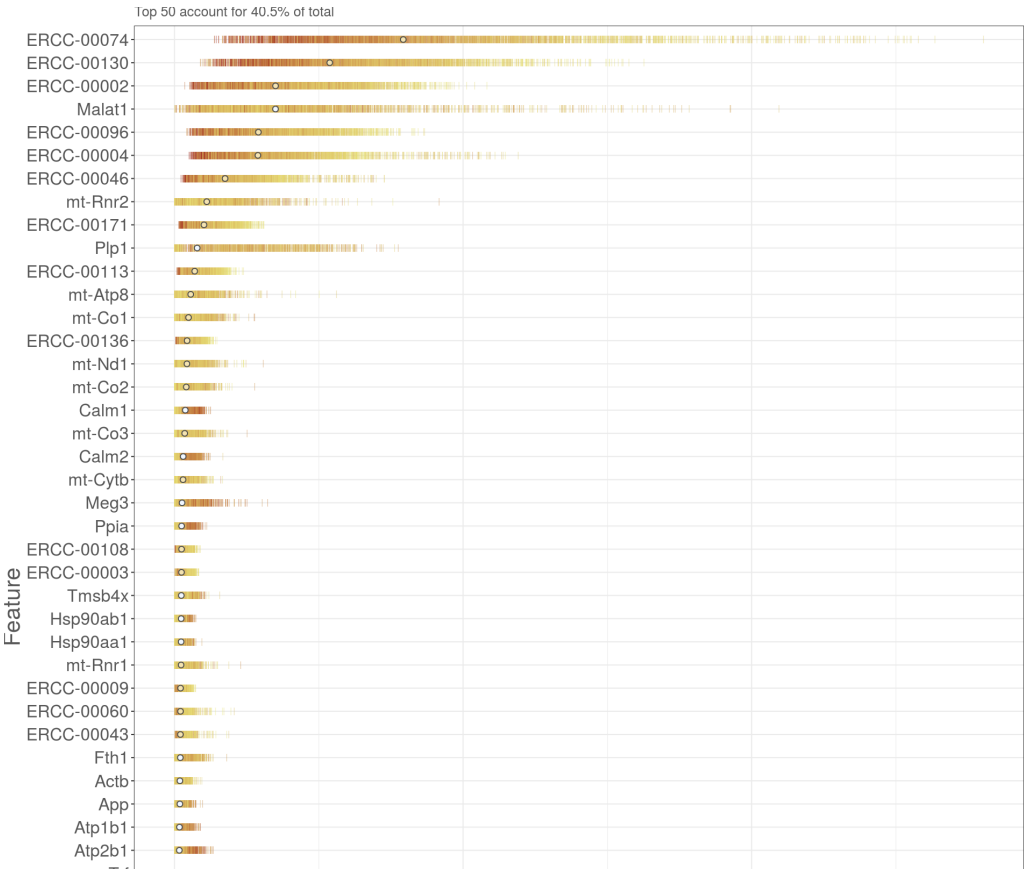
Figure 2: Cell cycle phase scores from applying the pair-based classifier on the brain dataset, where each point represents a cell.

However, the intepretation of this result requires some caution due to differences between the training and test datasets. The classifier was trained on C1 SMARTer data and accounts for the biases in that protocol. The brain dataset uses UMI counts, which has a different set of biases, e.g., 3'-end coverage only, no length bias, no amplification noise. Furthermore, many neuronal cell types are expected to lie in the G0 resting phase, which is distinct from the other phases of the cell cycle (Coller, Sang, and Roberts 2006). `cyclone` will generally assign such cells to the closest known phase in the training set, which would be G1.

5 Examining gene-level metrics

Figure 3 shows the most highly expressed genes across the cell population in the brain dataset. This is mostly occupied by spike-in transcripts, reflecting the use of spike-in concentrations that span the entire range of expression. There are also a number of constitutively expressed genes, as expected.

```
fontsize <- theme(axis.text=element_text(size=12), axis.titl
e=element_text(size=16))
plotHighestExprs(sce, n=50) + fontsize
```



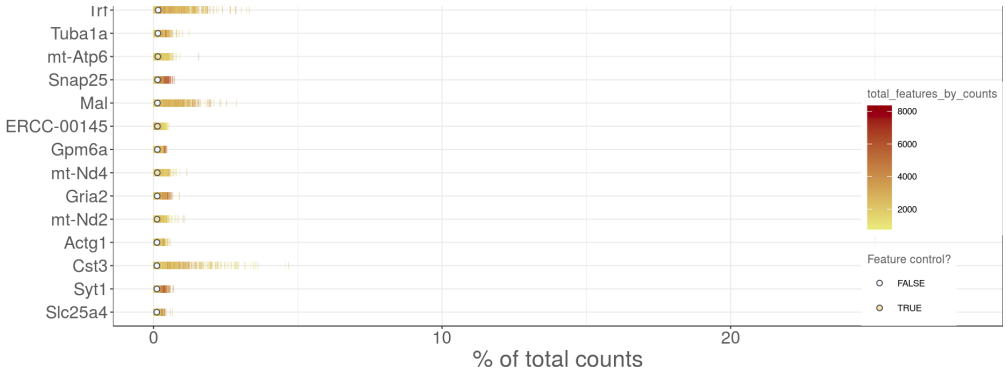


Figure 3: Percentage of total counts assigned to the top 50 most highly-abundant features in the brain dataset. For each feature, each bar represents the percentage assigned to that feature for a single cell, while the circle represents the average across all cells. Bars are coloured by the total number of expressed features in each cell, while circles are coloured according to whether the feature is labelled as a control feature.

Gene abundance is quantified by computing the average count across all cells (Figure 4). As previously mentioned, the UMI count is generally lower than the read count.

```
ave.counts <- calcAverage(sce, use_size_factors=FALSE)
hist(log10(ave.counts), breaks=100, main="", col="grey",
    xlab=expression(Log[10]~"average count"))
```

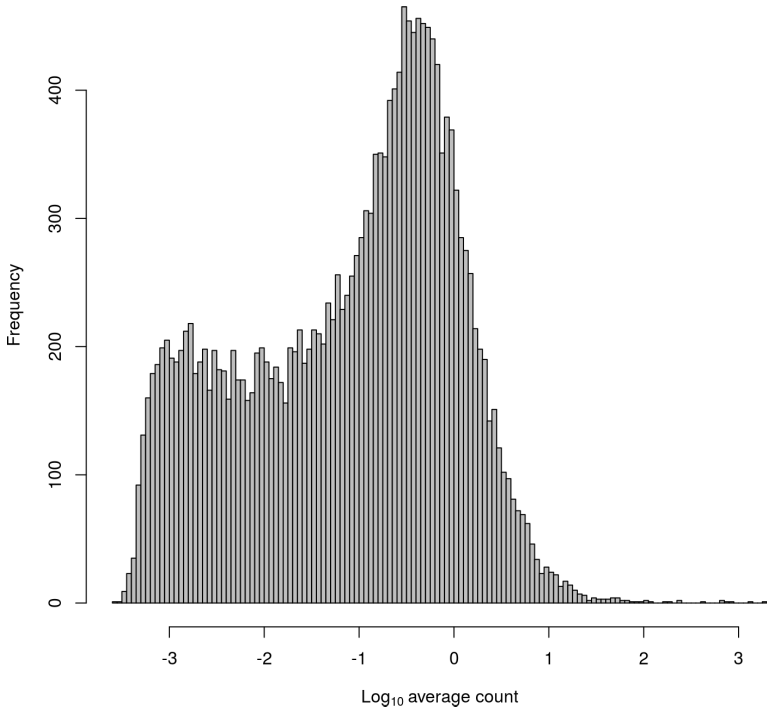


Figure 4: Histogram of log-average counts for all genes in the brain dataset.

We save the average counts into the `SingleCellExperiment` object for later use. We also remove genes that have average counts of zero, as this means that they are not expressed in any cell.

```
rowData(sce)$ave.count <- ave.counts
to.keep <- ave.counts > 0
sce <- sce[to.keep,]
summary(to.keep)
```

```
##    Mode   FALSE    TRUE
## logical       2   19894
```

6 Normalization of cell-specific biases

For endogenous genes, normalization is performed using the `computeSumFactors` function as previously described. Here, we cluster similar cells together and normalize the cells in each cluster using the deconvolution method (Lun, Bach, and Marioni 2016). This improves normalization accuracy by reducing the number of DE genes between cells in the same cluster. Scaling is then performed to ensure that size factors of cells in different clusters are comparable.

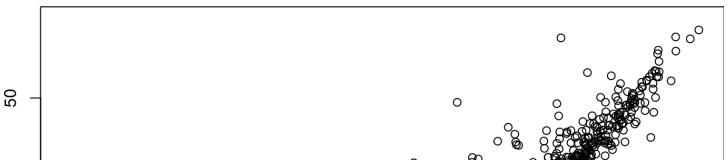
```
set.seed(1000)
clusters <- quickCluster(sce, min.mean=0.1, method="igraph")
sce <- computeSumFactors(sce, cluster=clusters, min.mean=0.
1)
summary(sizeFactors(sce))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.1255  0.4633  0.8207  1.0000  1.3366  4.7022
```

We use a average count threshold of 0.1 to define high-abundance genes to use during normalization. This is lower than the default threshold of `min.mean=1` in `computeSumFactors`, reflecting the fact that UMI counts are generally smaller than read counts.

Compared to the 416B analysis, more scatter is observed around the trend between the total count and size factor for each cell (Figure 5). This is consistent with an increased amount of DE between cells of different types, which compromises the accuracy of library size normalization (Robinson and Oshlack 2010). In contrast, the size factors are estimated based on median ratios and are more robust to the presence of DE between cells.

```
plot(sizeFactors(sce), sce$total_counts/1e3, log="xy",
    ylab="Library size (thousands)", xlab="Size factor")
```



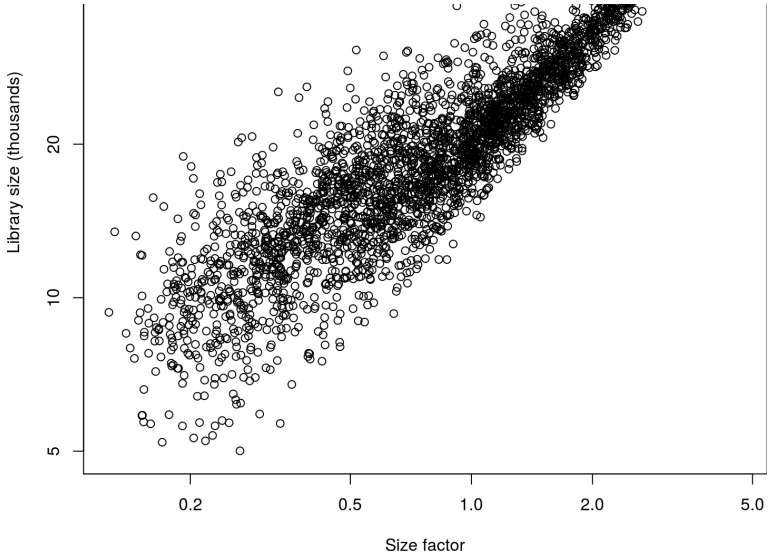


Figure 5: Size factors from deconvolution, plotted against library sizes for all cells in the brain dataset. Axes are shown on a log-scale.

We also compute size factors specific to the spike-in set, as previously described.

```
sce <- computeSpikeFactors(sce, type="Spike", general.use=FA
LSE)
```

Finally, normalized log-expression values are computed for each endogenous gene or spike-in transcript using the appropriate size factors.

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

Comments from Aaron:

- Only a rough clustering is required to avoid pooling together very different cell types in `computeSumFactors()`. This reduces the chance of violating the non-DE assumption that is made during any gene-based scaling normalization. We stress that there is no need for precise clustering at this step, as we will not be interpreting these clusters at all. Our normalization strategy is also robust to a moderate level of differential expression between cells in the same cluster, so careful definition of subclusters is not required. We do ask that there are sufficient cells in each cluster for pooling, which can be guaranteed with the `min.size=` argument in `quickCluster()`.
- `quickCluster` uses distances based on Spearman's rank correlation for clustering. This ensures that scaling biases in the counts do not affect clustering, but yields very coarse clusters and is not recommended for biological interpretation. However, for the purposes of breaking up distinct cell types, this is more than sufficient.
- For large datasets, using `method="igraph"` in `quickCluster`

will speed up clustering. This uses a graph-based clustering algorithm with a randomization step, hence the need for `set.seed()`. See `?buildSNNGraph` for more details.

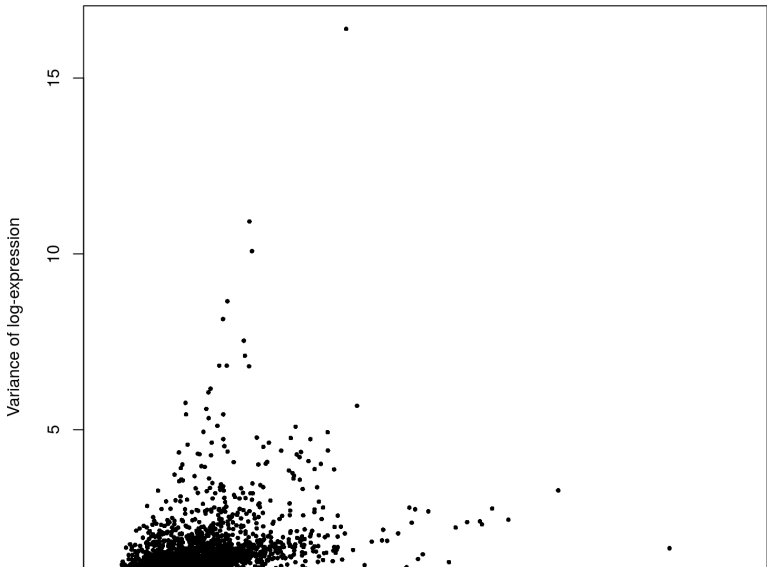
7 Modelling and removing technical noise

We model the technical noise by fitting a mean-variance trend to the spike-in transcripts, as previously described. In theory, we should block on the plate of origin for each cell. However, only 20-40 cells are available on each plate, and the population is also highly heterogeneous. This means that we cannot assume that the distribution of sampled cell types on each plate is the same. Thus, to avoid regressing out potential biology, we will not block on any factors in this analysis.

```
var.fit <- trendVar(sce, parametric=TRUE, loess.args=list(sp
an=0.4))
var.out <- decomposeVar(sce, var.fit)
```

Figure 6 indicates that the trend is fitted accurately to the technical variances. The technical and total variances are also much smaller than those in the 416B dataset. This is due to the use of UMIs, which reduces the noise caused by variable PCR amplification (Islam et al. 2014). Furthermore, the spike-in trend is consistently lower than the variances of the endogenous genes. This reflects the heterogeneity in gene expression across cells of different types.

```
plot(var.out$mean, var.out$total, pch=16, cex=0.6, xlab="Mea
n log-expression",
    ylab="Variance of log-expression")
points(var.out$mean[isSpike(sce)], var.out$total[isSpike(sc
e)], col="red", pch=16)
curve(var.fit$trend(x), col="dodgerblue", add=TRUE, lwd=2)
```



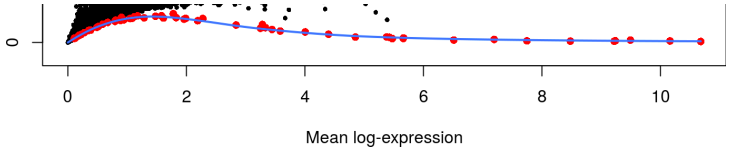


Figure 6: Variance of normalized log-expression values against the mean for each gene, calculated across all cells in the brain dataset after blocking on the sex effect. The blue line represents the mean-dependent trend in the technical variance of the spike-in transcripts (also highlighted as red points).

We check the distribution of expression values for the genes with the largest biological components to ensure that they are not driven by outliers (Figure 7). Some tweaking of the `plotExpression` parameters is necessary to visualize a large number of cells.

```
chosen.genes <- order(var.out$bio, decreasing=TRUE)[1:10]
plotExpression(sce, rownames(var.out)[chosen.genes],
    point_alpha=0.05, jitter_type="jitter") + fontsize
```