Characterizing gene expression variance in single-cell RNA-seq data

***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

Here, we describe some more detailed aspects of modelling variability in gene expression across the cell population. This includes the detection of significant highly variable genes (HVGs), as well as advanced modelling of the mean-variance trend in the presence of confounding factors.

2 Detecting highly variable genes

2.1 Overview

HVGs are defined as genes with biological components that are significantly greater than zero. These genes are interesting as they drive differences in the expression profiles between cells, and should be prioritized for further investigation. Formal detection of HVGs allows us to avoid genes that are highly variable due to technical factors such as sampling noise during RNA capture and library preparation. This adds another level of statistical rigour to our previous analyses, in which we only modelled the technical component.

2.2 Setting up the data

2.2.1 Loading the dataset

To demonstrate, we use data from haematopoietic stem cells (HSCs) (Wilson et al. 2015), generated using the Smart-seq2 protocol (Picelli et al. 2014) with ERCC spike-ins. Counts were obtained from NCBI GEO as a supplementary file using the accession number GSE61533 (http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE61533).

```
library(BiocFileCache)
bfc <- BiocFileCache("raw_data", ask=FALSE)
wilson.fname <- bfcrpath(bfc, file.path("ftp://ftp.ncbi.nlm.
nih.gov/geo/series",
    "GSE61nnn/GSE61533/suppl/GSE61533_HTSEQ_count_results.xl
s.gz"))

library(R.utils)
wilson.name2 <- "GSE61533_HTSEQ_count_results.xls"
gunzip(wilson.fname, destname=wilson.name2, remove=FALSE, ov
erwrite=TRUE)
```

Our first task is to load the count matrix into memory. In this case, some work is required to retrieve the data from the Gzip-compressed Excel format.

```
library(gdata)
all.counts <- read.xls(wilson.name2, sheet=1, header=TRUE)
rownames(all.counts) <- all.counts$ID
all.counts <- as.matrix(all.counts[,-1])
```

We store the results in a `SingleCellExperiment` object and identify the rows corresponding to the spike-ins based on the row names.

```
library(SingleCellExperiment)
sce.hsc <- SingleCellExperiment(list(counts=all.counts))
dim(sce.hsc)
```

```
## [1] 38498    96
```

```
is.spike <- grepl("^ERCC", rownames(sce.hsc))
isSpike(sce.hsc, "ERCC") <- is.spike
summary(is.spike)
```

```
##    Mode   FALSE    TRUE
## logical   38406      92
```

2.2.2 Quality control and normalization

For each cell, we calculate quality control metrics using the `calculateQCMetrics` function from *scater (https://bioconductor.org/packages/3.8/scater)* (McCarthy et al. 2017) as previously described. We filter out HSCs that are outliers for any metric, under the assumption that these represent low-quality libraries.

```
library(scater)
sce.hsc <- calculateQCMetrics(sce.hsc)
libsize.drop <- isOutlier(sce.hsc$total_counts, nmads=3, typ
e="lower", log=TRUE)
feature.drop <- isOutlier(sce.hsc$total_features_by_counts,
nmads=3, type="lower", log=TRUE)
spike.drop <- isOutlier(sce.hsc$pct_counts_ERCC, nmads=3, ty
pe="higher")
sce.hsc <- sce.hsc[,!(libsize.drop | feature.drop | spike.dr
op)]
data.frame(ByLibSize=sum(libsize.drop), ByFeature=sum(featur
e.drop),
    BySpike=sum(spike.drop), Remaining=ncol(sce.hsc))
```

```
##   ByLibSize ByFeature BySpike Remaining
## 1         2         2       3        92
```

We remove genes that are not expressed in any cell to reduce computational work in downstream steps.

```
to.keep <- nexprs(sce.hsc, byrow=TRUE) > 0
sce.hsc <- sce.hsc[to.keep,]
summary(to.keep)
```

```
##    Mode   FALSE    TRUE
## logical   17229   21269
```

We apply the deconvolution method (Lun, Bach, and Marioni 2016) to compute size factors for the endogenous genes (Lun, Bach, and Marioni 2016). Separate size factors for the spike-in transcripts are also calculated, as previously discussed. We then calculate log-transformed normalized expression values for further use.

```
library(scran)
sce.hsc <- computeSumFactors(sce.hsc)
summary(sizeFactors(sce.hsc))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.4075  0.8058  0.9604  1.0000  1.1503  2.0414
```

```
sce.hsc <- computeSpikeFactors(sce.hsc, type="ERCC", genera
l.use=FALSE)
summary(sizeFactors(sce.hsc, "ERCC"))
```

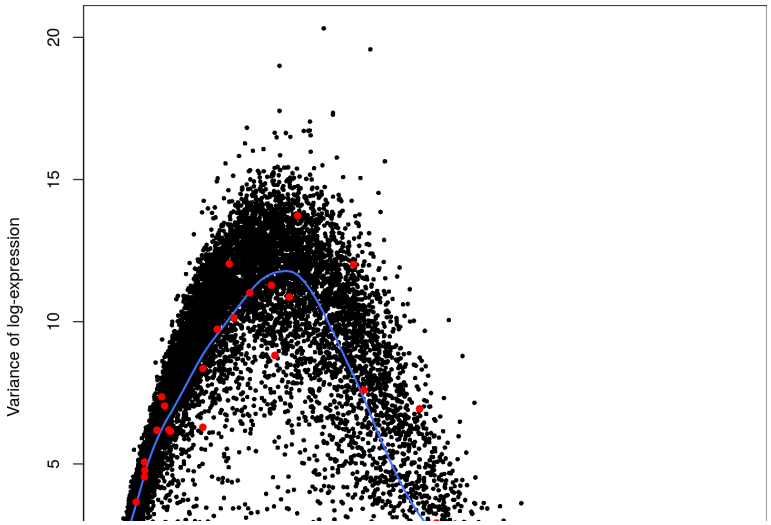
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.2562  0.6198  0.8623  1.0000  1.2122  3.0289
```

```
sce.hsc <- normalize(sce.hsc)
```

2.3 Testing for significantly positive biological components

We fit a mean-variance trend to the spike-in transcripts to quantify the technical component of the variance, as previously described. The biological component for each gene is defined as the difference between its total variance and the fitted value of the trend (Figure 1).

```
var.fit <- trendVar(sce.hsc, parametric=TRUE, loess.args=lis
t(span=0.3))
var.out <- decomposeVar(sce.hsc, var.fit)
plot(var.out$mean, var.out$total, pch=16, cex=0.6, xlab="Mea
n log-expression",
    ylab="Variance of log-expression")
curve(var.fit$trend(x), col="dodgerblue", lwd=2, add=TRUE)
cur.spike <- isSpike(sce.hsc)
points(var.out$mean[cur.spike], var.out$total[cur.spike], co
l="red", pch=16)
```



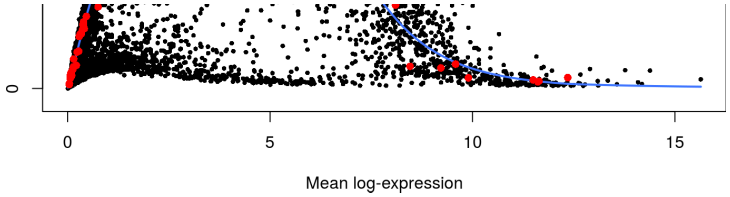


Figure 1: Variance of normalized log-expression values for each gene in the HSC dataset, plotted against the mean log-expression. The blue line represents the mean-dependent trend fitted to the variances of the spike-in transcripts (red).

We define HVGs as those genes that have a biological component that is significantly greater than zero. We use a false discovery rate (FDR) of 5% after correcting for multiple testing with the Benjamini-Hochberg method.

```
hvg.out <- var.out[which(var.out$FDR <= 0.05),]
nrow(hvg.out)
```

```
## [1] 511
```

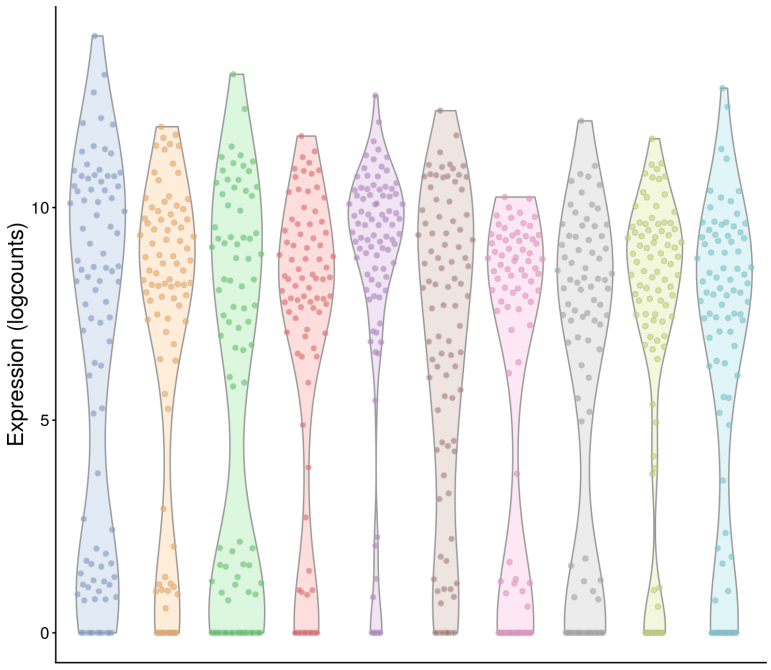
We rank the results to focus on genes with larger biological components. This highlights an interesting aspect of the underlying hypothesis test, which is based on the ratio of the total variance to the expected technical variance. Ranking based on *p*-value tends to prioritize HVGs that are more likely to be true positives but, at the same time, less likely to be interesting. This is because the ratio can be very large for HVGs that have very low total variance and do not contribute much to the cell-cell heterogeneity.

```
hvg.out <- hvg.out[order(hvg.out$bio, decreasing=TRUE),]
write.table(file="hsc_hvg.tsv", hvg.out, sep="\t", quote=FAL
SE, col.names=NA)
head(hvg.out)
```

```
## DataFrame with 6 rows and 6 columns
##                      mean            total              b
io             tech
##                 <numeric>        <numeric>        <numeri
c>        <numeric>
## Fos      6.46229730105568 19.5774140745205 12.82210028722
27 6.75531378729785
## Dusp1    6.82314467202067 15.6360357802411 10.11622902093
77 5.51980675930337
## Rgs1     5.31345464519062 20.3107030137294 10.01194508536
34 10.2987579283659
## Ppp1r15a  6.6657972702732 14.5266651233921 8.475969311978
98 6.05069581141309
## Ly6a     8.40354443081163 10.0583341434802 8.058001010825
91 2.00033313265429
## Egr1     6.71592505529894 13.8570278172783 7.977527239622
34 5.87950057765595
##                       p.value                  FDR
##                     <numeric>            <numeric>
## Fos      1.01066135261343e-18 7.14571265009451e-16
## Dusp1    7.24908881759115e-18 4.15568710567367e-15
## Rgs1     9.43425347577711e-08 1.10557983687684e-05
## Ppp1r15a  1.7343225455171e-12 4.90489540172842e-10
## Ly6a     2.99530585166056e-50 9.07620463136746e-47
## Egr1     5.70569033977291e-12 1.49411602218424e-09
```

We check the distribution of expression values for the genes with the largest biological components. This ensures that the variance estimate is not driven by one or two outlier cells (Figure 2).

```
fontsize <- theme(axis.text=element_text(size=12), axis.titl
e=element_text(size=16))
plotExpression(sce.hsc, features=rownames(hvg.out)[1:10]) +
fontsize
```



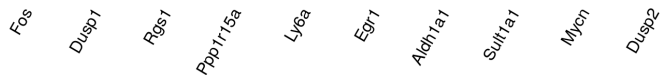


Figure 2: Violin plots of normalized log-expression values for the top 10 genes with the largest biological components in the HSC dataset. Each point represents the log-expression value in a single cell.

2.4 Further comments

There are many other strategies for defining HVGs, based on a variety of metrics:

- the coefficient of variation, using the `technicalCV2()` function (Brennecke et al. 2013) or the `DM()` function (Kim et al. 2015) in *scran (https://bioconductor.org/packages/3.8/scran)*.
- the dispersion parameter in the negative binomial distribution, using the `estimateDisp()` function in *edgeR (https://bioconductor.org/packages/3.8/edgeR)* (McCarthy, Chen, and Smyth 2012).
- a proportion of total variability, using methods in the *BASiCS (https://bioconductor.org/packages/3.8/BASiCS)* package (Vallejos, Marioni, and Richardson 2015).

Here, we used the variance of the log-expression values because the log-transformation protects against genes with strong expression in only one or two cells. This ensures that the set of top HVGs is not dominated by genes with (mostly uninteresting) outlier expression patterns.

We also save the HSC dataset to file for later use, using `saveRDS()` as previously described.

```
saveRDS(sce.hsc, file="hsc_data.rds")
```

3 Fine-tuning the mean-variance trend fit

3.1 Details of trend fitting parameters

Fitting of the mean-variance trend with `trendVar()` is complicated by the small number of spike-in transcripts and the uneven distribution of their abundances. For low numbers of cells, these issues are exacerbated by the low precision of the variance estimates. Thus, some tuning of trend parameters such as `span` may be required to achieve a suitable fit - see `?trendVar` for more details. Setting `parametric=TRUE` is especially useful for modelling the expected wave-like shape of the mean-variance relationship. (This is not the default setting as it is not robust for arbitrary trend shapes.)

The `trendVar` function will also automatically filter out low-

abundance genes prior to trend fitting. This ensures that low-abundance genes do not interfere with the fit due to discreteness, which biases the estimate of variability of the variances around the trend; or due to the frequency of low-abundance genes, which reduces the sensitivity of span-based smoothing algorithms at higher abundances. The internal choice of filtering strategy involves a number of considerations:

- Filtering uses the average of log-expression values rather than the (library size-adjusted) average count. The mean log-expression is independent of the variance estimate in a linear modelling framework (Bourgon, Gentleman, and Huber 2010), which ensures that the filter does not introduce spurious trends in the variances at the filter boundary.
- The filter threshold is specified with the `min.mean` argument in `trendVar`. We use the default threshold of 0.1 (`min.mean`) based on the appearance of discrete patterns in the variance estimates for simulated Poisson-distributed counts. Lower thresholds of 0.001-0.01 may be more suitable for very sparse data, e.g., from droplet-based protocols.
- The filter used in `trendVar` is *not* applied in `decomposeVar` by default. Retention of all genes ensures that weak biological signal from rare subpopulations is not discarded. To apply the filter in `decomposeVar`, users should set `subset.row=rowMeans(logcounts(sce)) > 0.1` in the function call.

Comments from Aaron:

- On occasion, users may observe a warning from `trendVar()` about the lack of centering in the size factors. Recall that the trend is fitted to the mean and variances of the spike-in transcripts, and the technical component for each *endogenous* gene is estimated by interpolation. This assumes that an endogenous gene is comparable to a spike-in transcript of the same abundance. In particular, we assume that variation is primarily driven by the magnitude of the counts, based on the well-known mean-variance relationships in count models. Thus, we need to ensure that similarities in the average counts are preserved in the normalized expression values. This is achieved by centering the gene- and spike-in-based size factors in `normalize()`, such that features with similar average counts will also have similar normalized abundances. However, if the `SingleCellExperiment` object was manipulated (e.g., subsetted) *after* `normalize()` and *before* `trendVar()`, centering may not be preserved - hence the warning.

3.2 When spike-ins are unavailable

In some datasets, spike-in RNA may not have been added in

appropriate quantities (or indeed at all). It may also be inappropriate to assume Poisson technical noise with `makeTechTrend()`, especially for read count data where amplification noise is non-negligible. In such cases, an alternative approach is to fit the trend to the variance estimates of the endogenous genes. This is done using the `use.spikes=FALSE` setting in `trendVar`, as shown below for the HSC dataset.

```
var.fit.nospike <- trendVar(sce.hsc, parametric=TRUE,
    use.spikes=FALSE, loess.args=list(span=0.2))
var.out.nospike <- decomposeVar(sce.hsc, var.fit.nospike)
```

The simplest interpretation of the results assumes that the majority of genes are not variably expressed. This means that the technical component dominates the total variance for most genes, such that the fitted trend can be treated as an estimate of the technical component. In Figure 3, the trend passes through or close to most of the spike-in variances, indicating that our assumption is valid.

```
plot(var.out.nospike$mean, var.out.nospike$total, pch=16, ce
x=0.6,
    xlab="Mean log-expression", ylab="Variance of log-expres
sion")
curve(var.fit.nospike$trend(x), col="dodgerblue", lwd=2, add
=TRUE)
points(var.out.nospike$mean[cur.spike], var.out.nospike$tota
l[cur.spike], col="red", pch=16)
```

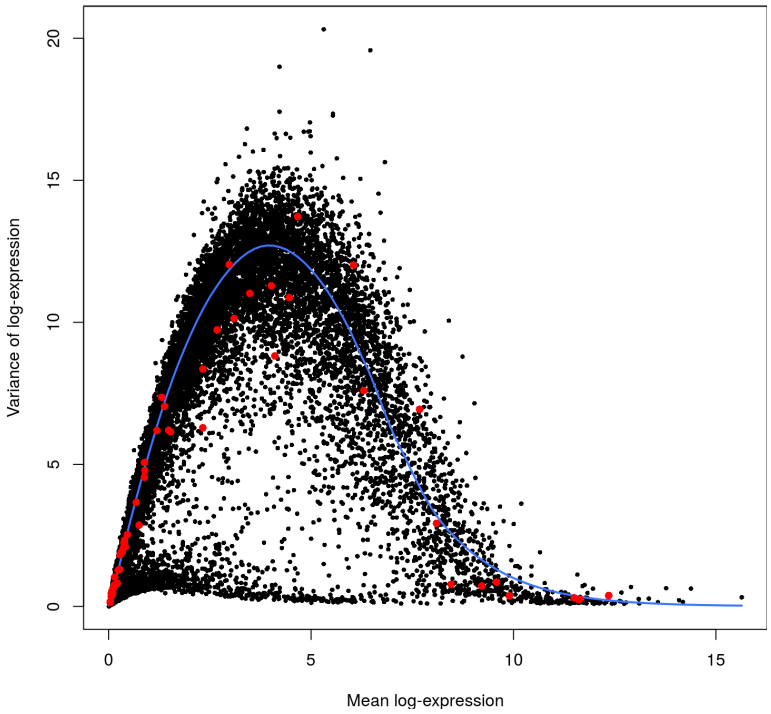


Figure 3: Variance of normalized log-expression values for each gene in the 416B dataset, plotted against the mean log-expression. The blue line represents the mean-dependent trend fitted to the variances of the endogenous genes (black),

with spike-in transcripts shown in red.

If our assumption does not hold, the output of decomposeVar is more difficult to interpret. The fitted value of the trend can no longer be generally interpreted as the technical component, as it contains some biological variation as well. Instead, recall that the biological component reported by `decomposeVar` represents the residual for each gene over the majority of genes with the same abundance. One could assume that the variabilities of most genes are driven by constitutive “house-keeping” processes, which are biological in origin but generally uninteresting. Any gene with an increase in its variance is *relatively* highly variable and can be prioritized for further study.

4 Blocking on uninteresting factors of variation

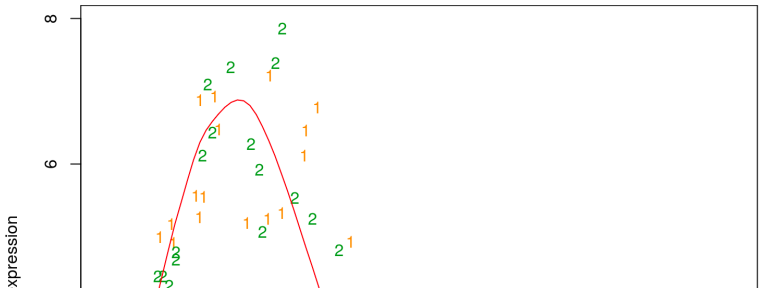
4.1 Using the `block=` argument

Our previous analysis of the 416B dataset specified `block=` in `trendVar()` to ensure that systematic differences between plates do not inflate the variance. This involves estimating the mean and variance of the log-expression *separately* in each plate, followed by fitting a single trend to the plate-specific means and variances of all spike-in transcripts. In doing so, we implicitly assume that the trend is the same between plates, which is reasonable for this dataset (Figure 4).

```
# Loading the saved object.
sce.416B <- readRDS("416B_data.rds")

# Repeating the trendVar() call.
var.fit <- trendVar(sce.416B, parametric=TRUE, block=sce.416
B$Plate,
    loess.args=list(span=0.3))

matplot(var.fit$means, var.fit$vars, col=c("darkorange", "fo
restgreen"),
    xlab="Mean log-expression", ylab="Variance of log-expres
sion")
curve(var.fit$trend(x), add=TRUE, col="red")
```



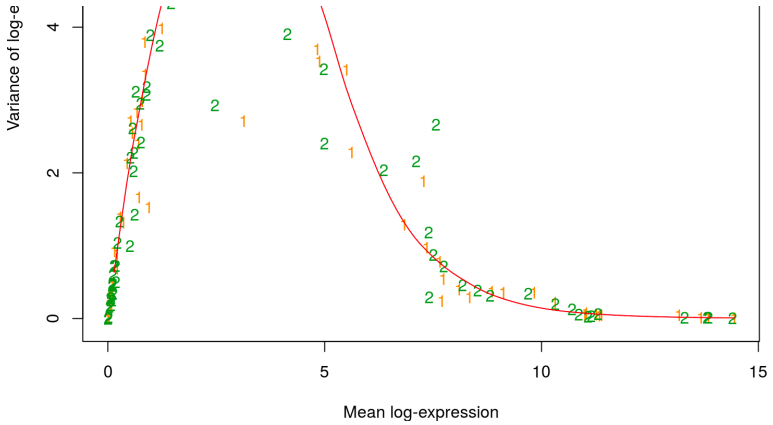


Figure 4: Plate-specific variance estimates for all spike-in transcripts in the 416B dataset, plotted against the plate-specific means. Each point represents a spike-in transcript, numbered by the plate from which the values were estimated. The red line denotes the fitted mean-variance trend.

The use of `block=` also assumes that the average size factor within each plate is close to unity for both endogenous genes and spike-in transcripts. This ensures that scaling normalization preserves the magnitude of the counts, allowing genes of the same average abundance to be compared within and across plates. Here, the distributions of size factors exhibit only modest deviations from unity in the averages for the endogenous genes and spike-in transcripts (Figure 5), indicating that our assumption is again reasonable.

```
tmp.416B <- sce.416B
tmp.416B$log_size_factor <- log(sizeFactors(sce.416B))
tmp.416B$log_size_factor_ERCC <- log(sizeFactors(sce.416B, "
ERCC"))
p1 <- plotColData(tmp.416B, x="Plate", y="log_size_factor")
p2 <- plotColData(tmp.416B, x="Plate", y="log_size_factor_ER
CC")
multiplot(p1, p2, cols=2)
```

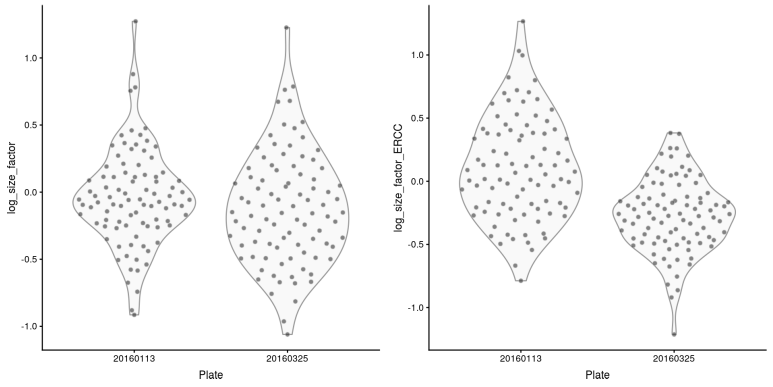


Figure 5: Plate-specific distribution of the size factors for endogenous genes (left) and spike-in transcripts (right).

The most obvious violation of the above assumption occurs when more spike-in RNA is added in a particular batch. Spike-in size factors would then be systematically larger than unity in that

batch, meaning that average abundances after normalization would not be comparable between spike-in transcripts and endogenous genes. This would compromise the accuracy of estimation of the technical component by interpolation from the trend. More generally, batches that are processed separately may not exhibit to the same technical variation, such that the use of a single trend would be inappropriate.

4.2 Fitting batch-specific trends

For datasets containing multiple batches, an alternative strategy is to perform trend fitting and variance decomposition separately for each batch. This accommodates differences in the mean-variance trends between batches, especially if a different amount of spike-in RNA was added to the cells in each batch. We demonstrate this approach by treating each plate in the 416B dataset as a different batch, using the `multiBlockVar()` function. This yields plate-specific estimates of the biological and technical components for each gene.

```
sce.416B.2 <- multiBlockNorm(sce.416B, sce.416B$Plate)
comb.out <- multiBlockVar(sce.416B.2, block=sce.416B.2$Plat
e,
    trend.args=list(parametric=TRUE, loess.args=list(span=0.
4)))
```

Statistics are combined across multiple batches using the `combineVar()` function within `multiBlockVar()`. This function computes a weighted average across batches for the means and variances, and applies Fisher's method for combining the *p*-values. These results can be used in downstream functions such as `denoisePCA`, or for detecting highly variable genes (see below).

```
head(comb.out[,1:6])
```

```
## DataFrame with 6 rows and 6 columns
##                                    mean                tota
l
##                               <numeric>           <numeric
>
## ENSMUSG00000103377 0.00807160215879455  0.011921865484604
6
## ENSMUSG00000103147  0.0346526071354525  0.072219615904211
1
## ENSMUSG00000103161 0.00519472220098498 0.0049385769482036
2
## ENSMUSG00000102331  0.0186660929969477  0.032923591717629
1
## ENSMUSG00000102948  0.0590569999301664  0.088137125286206
4
## Rp1                 0.0970243710886109   0.45233813512634
9
##                                   bio                tech
p.value
##                               <numeric>           <numeric>
<numeric>
## ENSMUSG00000103377 -0.0277229724633449 0.0396448379479495
1
## ENSMUSG00000103147 -0.0764556945917083  0.148675310495919
0.999999999962404
## ENSMUSG00000103161 -0.0173491250826678 0.0222877020308714
1
## ENSMUSG00000102331 -0.0540089342569103 0.0869325259745394
0.999999999999991
## ENSMUSG00000102948  -0.169989420230668  0.258126545516874
1
## Rp1                0.00813766950901805  0.444200465617331
0.163473096868316
##                                    FDR
##                               <numeric>
## ENSMUSG00000103377                    1
## ENSMUSG00000103147                    1
## ENSMUSG00000103161                    1
## ENSMUSG00000102331                    1
## ENSMUSG00000102948                    1
## Rp1                0.538786122896226
```

We visualize the quality of the batch-specific trend fits by extracting the relevant statistics from `comb.out` (Figure 6).

```
par(mfrow=c(1,2))
is.spike <- isSpike(sce.416B.2)
for (plate in levels(sce.416B.2$Plate)) {
    cur.out <- comb.out$per.block[[plate]]
    plot(cur.out$mean, cur.out$total, pch=16, cex=0.6, xlab
="Mean log-expression",
        ylab="Variance of log-expression", main=plate)
    curve(metadata(cur.out)$trend(x), col="dodgerblue", lwd=
2, add=TRUE)
    points(cur.out$mean[is.spike], cur.out$total[is.spike],
col="red", pch=16)
}
```

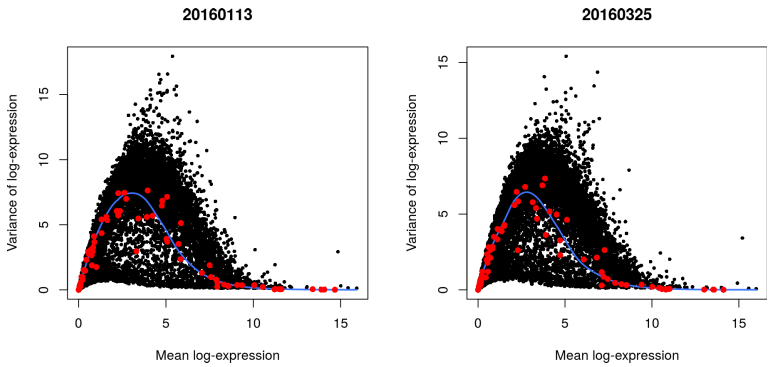


Figure 6: Variance of normalized log-expression values for each gene in each plate of the 416B dataset, plotted against the mean log-expression. The blue line represents the mean-dependent trend fitted to the variances of the spike-in transcripts (red).

By fitting separate trends, we avoid the need to assume that a single trend is present across batches. However, this also reduces the precision of each trend fit, as less information is available within each batch. We recommend using `block=` as the default unless there is clear evidence for differences in the trends between batches.

Comments from Aaron:

- We run `multiBlockNorm()` to adjust the size factors within each level of the blocking factor. This ensures that the mean of the spike-in size factors across cells in each batch is equal to that of the gene-based size factors. Log-normalized expression values are then recalculated using these centred size factors. This procedure ensures that the average abundances of the spike-in transcripts are comparable to the endogenous genes, avoiding problems due to differences in the quantity of spike-in RNA between batches. Otherwise, if the globally-centred size factors (from `normalize()`) were used, there would be a systematic difference in the scaling of spike-in transcripts compared to endogenous genes in batches with more or less spike-in RNA than the dataset average. The fitted trend would then be shifted along the x-axis and fail to accurately capture the technical

component for each gene.

4.3 Using the `design=` argument

For completeness, it is worth mentioning the `design=` argument in `trendVar()`. This will estimate the residual variance from a linear model fitted to the log-normalized expression values for each gene. The linear model can include blocking factors for known unwanted factors of variation, ensuring that they do not inflate the variance estimate. The technical component for each gene is obtained at the average abundance across all cells.

```
lfit <- trendVar(sce.416B, design=model.matrix(~sce.416B$Pla
te))
```

We do not recommend using this approach for categorical blocking factors in one-way layouts. This is because it does not consider the mean of each blocking level, resulting in an inaccurate estimate of the technical component in the presence of a strong blocking effect. However, it is the only choice for dealing with real covariates or multiple blocking factors in an additive model.

5 Concluding remarks

All software packages used in this workflow are publicly available from the Comprehensive R Archive Network (https://cran.r-project.org (https://cran.r-project.org)) or the Bioconductor project (http://bioconductor.org (http://bioconductor.org)). The specific version numbers of the packages used are shown below, along with the version of the R installation.

```
sessionInfo()
```

```
## R version 3.5.2 (2018-12-20)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 16.04.5 LTS
##
## Matrix products: default
## BLAS: /home/biocbuild/bbs-3.8-bioc/R/lib/libRblas.so
## LAPACK: /home/biocbuild/bbs-3.8-bioc/R/lib/libRlapack.so
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices uti
ls     datasets
## [8] methods   base
##
## other attached packages:
##  [1] gdata_2.18.0
##  [2] R.utils_2.7.0
##  [3] R.oo_1.22.0
##  [4] R.methodsS3_1.7.1
##  [5] scRNAseq_1.8.0
##  [6] edgeR_3.24.3
##  [7] Matrix_1.2-15
##  [8] org.Hs.eg.db_3.7.0
##  [9] EnsDb.Hsapiens.v86_2.99.0
## [10] ensembldb_2.6.3
## [11] AnnotationFilter_1.6.0
## [12] DropletUtils_1.2.2
## [13] pheatmap_1.0.12
## [14] cluster_2.0.7-1
## [15] dynamicTreeCut_1.63-1
## [16] limma_3.38.3
## [17] scran_1.10.2
## [18] scater_1.10.1
## [19] ggplot2_3.1.0
## [20] TxDb.Mmusculus.UCSC.mm10.ensGene_3.4.0
## [21] GenomicFeatures_1.34.1
## [22] org.Mm.eg.db_3.7.0
## [23] AnnotationDbi_1.44.0
## [24] SingleCellExperiment_1.4.1
## [25] SummarizedExperiment_1.12.0
## [26] DelayedArray_0.8.0
## [27] BiocParallel_1.16.5
## [28] matrixStats_0.54.0
## [29] Biobase_2.42.0
## [30] GenomicRanges_1.34.0
## [31] GenomeInfoDb_1.18.1
## [32] IRanges_2.16.0
```

```
## [33] S4Vectors_0.20.1
## [34] BiocGenerics_0.28.0
## [35] bindrcpp_0.2.2
## [36] BiocFileCache_1.6.0
## [37] dbplyr_1.2.2
## [38] knitr_1.21
## [39] BiocStyle_2.10.0
##
## loaded via a namespace (and not attached):
##   [1] tidyselect_0.2.5         RSQLite_2.1.1
##   [3] grid_3.5.2               trimcluster_0.1-2.1
##   [5] Rtsne_0.15               munsell_0.5.0
##   [7] statmod_1.4.30           sROC_0.1-2
##   [9] withr_2.1.2              colorspace_1.3-2
##  [11] highr_0.7                robustbase_0.93-3
##  [13] vcd_1.4-4                VIM_4.7.0
##  [15] labeling_0.3             GenomeInfoDbData_1.2.0
##  [17] cvTools_0.3.2            bit64_0.9-7
##  [19] rhdf5_2.26.2             xfun_0.4
##  [21] diptest_0.75-7           R6_2.3.0
##  [23] ggbeeswarm_0.6.0         robCompositions_2.0.9
##  [25] locfit_1.5-9.1           mvoutlier_2.0.9
##  [27] flexmix_2.3-14           bitops_1.0-6
##  [29] reshape_0.8.8            assertthat_0.2.0
##  [31] scales_1.0.0             nnet_7.3-12
##  [33] beeswarm_0.2.3           gtable_0.2.0
##  [35] rlang_0.3.1              splines_3.5.2
##  [37] rtracklayer_1.42.1       lazyeval_0.2.1
##  [39] BiocManager_1.30.4       yaml_2.2.0
##  [41] reshape2_1.4.3           abind_1.4-5
##  [43] tools_3.5.2              bookdown_0.9
##  [45] zCompositions_1.1.2      RColorBrewer_1.1-2
##  [47] Rcpp_1.0.0               plyr_1.8.4
##  [49] progress_1.2.0           zlibbioc_1.28.0
##  [51] purrr_0.2.5              RCurl_1.95-4.11
##  [53] prettyunits_1.0.2        viridis_0.5.1
##  [55] cowplot_0.9.4            zoo_1.8-4
##  [57] haven_2.0.0              magrittr_1.5
##  [59] data.table_1.11.8        openxlsx_4.1.0
##  [61] lmtest_0.9-36            truncnorm_1.0-8
##  [63] mvtnorm_1.0-8            ProtGenerics_1.14.0
##  [65] hms_0.4.2                evaluate_0.12
##  [67] XML_3.98-1.16            rio_0.5.16
##  [69] mclust_5.4.2             readxl_1.2.0
##  [71] gridExtra_2.3            compiler_3.5.2
##  [73] biomaRt_2.38.0           tibble_2.0.0
##  [75] KernSmooth_2.23-15       crayon_1.3.4
##  [77] htmltools_0.3.6          pcaPP_1.9-73
##  [79] rrcov_1.4-7              DBI_1.0.0
##  [81] MASS_7.3-51.1            fpc_2.1-11.1
##  [83] rappdirs_0.3.1           boot_1.3-20
##  [85] car_3.0-2                sgeostat_1.0-27
##  [87] bindr_0.1.1              igraph_1.2.2
##  [89] forcats_0.3.0            pkgconfig_2.0.2
##  [91] GenomicAlignments_1.18.1 foreign_0.8-71
```

```
##  [93] laeken_0.4.6            sp_1.3-1
##  [95] vipor_0.4.5             XVector_0.22.0
##  [97] NADA_1.6-1              stringr_1.3.1
##  [99] digest_0.6.18           pls_2.7-0
## [101] Biostrings_2.50.2       rmarkdown_1.11
## [103] cellranger_1.1.0        DelayedMatrixStats_1.4.0
## [105] curl_3.2                kernlab_0.9-27
## [107] gtools_3.8.1            Rsamtools_1.34.0
## [109] modeltools_0.2-22       Rhdf5lib_1.4.2
## [111] carData_3.0-2           BiocNeighbors_1.0.0
## [113] viridisLite_0.3.0       pillar_1.3.1
## [115] lattice_0.20-38         GGally_1.4.0
## [117] httr_1.4.0              DEoptimR_1.0-8
## [119] survival_2.43-3         glue_1.3.0
## [121] zip_1.0.0               prabclus_2.2-6
## [123] bit_1.1-14              class_7.3-15
## [125] stringi_1.2.4           HDF5Array_1.10.1
## [127] blob_1.1.1              memoise_1.1.0
## [129] dplyr_0.7.8             irlba_2.3.2
## [131] e1071_1.7-0
```

References

Bourgon, R., R. Gentleman, and W. Huber. 2010. “Independent filtering increases detection power for high-throughput experiments.” *Proc. Natl. Acad. Sci. U.S.A.* 107 (21):9546–51.

Brennecke, P., S. Anders, J. K. Kim, A. A. Kołodziejczyk, X. Zhang, V. Proserpio, B. Baying, et al. 2013. “Accounting for technical noise in single-cell RNA-seq experiments.” *Nat. Methods* 10 (11):1093–5.

Kim, J. K., A. A. Kołodziejczyk, T. Illicic, S. A. Teichmann, and J. C. Marioni. 2015. “Characterizing noise structure in single-cell RNA-seq distinguishes genuine from technical stochastic allelic expression.” *Nat. Commun.* 6:8687.

Lun, A. T., K. Bach, and J. C. Marioni. 2016. “Pooling across cells to normalize single-cell RNA sequencing data with many zero counts.” *Genome Biol.* 17 (April):75.

McCarthy, D. J., K. R. Campbell, A. T. Lun, and Q. F. Wills. 2017. “Scater: pre-processing, quality control, normalization and visualization of single-cell RNA-seq data in R.” *Bioinformatics* 33 (8):1179–86.

McCarthy, D. J., Y. Chen, and G. K. Smyth. 2012. “Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation.” *Nucleic Acids Res.* 40 (10):4288–97.

Picelli, S., O. R. Faridani, A. K. Bjorklund, G. Winberg, S. Sagasser, and R. Sandberg. 2014. “Full-length RNA-seq from single cells

using Smart-seq2." *Nat Protoc* 9 (1):171–81.

Vallejos, C. A., J. C. Marioni, and S. Richardson. 2015. "BASiCS: Bayesian analysis of single-cell sequencing data." *PLoS Comput. Biol.* 11 (6):e1004333.

Wilson, N. K., D. G. Kent, F. Buettner, M. Shehata, I. C. Macaulay, F. J. Calero-Nieto, M. Sanchez Castillo, et al. 2015. "Combined single-cell functional and gene expression analysis resolves heterogeneity within stem cell populations." *Cell Stem Cell* 16 (6):712–24.