Accounting for technical noise in single-cell RNA-seq experiments – Supplement II

All analyses in the paper were carried out in the statistical programming language *R*. This Supplement documents the complete workflow to recreate all figures and numbers shown in the paper. It is extensively commented in order to demonstrate how our suggested analysis method is performed in practice and facilitate re-implementation by the user.

Table of Contents

- 1 Preparations
  - 1.1 Packages
  - 1.2 Count table
- 2 Analysis of the GL2 cell data
  - 2.1 Preparing the count table
  - 2.2 Normalization
  - 2.3 Plot of normalized counts
  - 2.4 Estimating technical noise
    - 2.4.1 Plot of the fit
  - 2.5 Testing plant genes for high variance
    - 2.5.1 Plot
    - 2.5.2 Table of highly variable genes
  - 2.6 GO analysis
    - 2.6.1 Results:
  - 2.7 Heatmap
  - 2.8 Read downsampling
  - 2.9 Effect of transcript length
- 3 Analysis of the QC cells
  - 3.1 Preparing the count table.
  - 3.2 Normalization
  - 3.3 Plot of normalized counts
  - 3.4 Estimating technical noise
    - 3.4.1 Plot of the fit
  - 3.5 Testing plant genes for high variance
    - 3.5.1 Plot
  - 3.6 Table of highly variable genes
  - 3.7 GO analysis
    - 3.7.1 Results:
  - 3.8 Heatmap
- 4 Analysis of the mouse cells
  - 4.1 Count table and normalization
  - 4.2 Fit technical noise
  - 4.3 Test for high variance
  - 4.4 A diagnostic plot
  - 4.5 Plot of results

1 Preparations

1.1 Packages

We load all R and Bioconductor packages that we need for this analysis:

```
library( DESeq )
library( genefilter )
library( EBImage )
library( statmod )
library( topGO )
library( org.At.tair.db )
options( max.print=300, width=100 )
```

The sessionInfo command gives the versions of R and all packages used in the present analysis run:

```
sessionInfo()
```

```
R version 2.15.3 (2013-03-01)
Platform: x86_64-pc-linux-gnu (64-bit)

locale:
 [1] LC_CTYPE=en_GB.UTF-8       LC_NUMERIC=C                 LC_TIME=en_GB.UTF-8
 [4] LC_COLLATE=en_GB.UTF-8     LC_MONETARY=en_GB.UTF-8    LC_MESSAGES=en_GB.UTF-8
 [7] LC_PAPER=C                 LC_NAME=C                  LC_ADDRESS=C
[10] LC_TELEPHONE=C             LC_MEASUREMENT=en_GB.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
 [1] org.At.tair.db_2.8.0 topGO_2.10.0         SparseM_0.96         GO.db_2.8.0
 [5] RSQLite_0.11.2       DBI_0.2-5            AnnotationDbi_1.20.3 graph_1.36.2
 [9] statmod_1.4.17       EBImage_4.0.0        genefilter_1.40.0    DESeq_1.11.6
[13] lattice_0.20-13      locfit_1.5-8         Biobase_2.18.0       BiocGenerics_0.4.0

loaded via a namespace (and not attached):
 [1] abind_1.4-0        annotate_1.36.0    geneplotter_1.36.0 grid_2.15.3        IRanges_1.
 [6] jpeg_0.1-2         parallel_2.15.3    png_0.1-4          RColorBrewer_1.0-5 splines_2.
[11] stats4_2.15.3      survival_2.36-14   tiff_0.1-3         tools_2.15.3       XML_3.95-0
[16] xtable_1.7-0
```

1.2 Count table

Supplementary Table 8 contains the raw read counts for the seven GL2 samples and the six QC samples. We save this table as a CSV file and load it into R.

```
fullCountTable <- read.csv( "Supplementary_Table_8.csv", header=TRUE, row.names=1 )
head( fullCountTable )
```

```
          QC1 QC2  QC3 QC4 QC5 QC6 GL2.1 GL2.2 GL2.3 GL2.4 GL2.5 GL2.6 GL2.7
AT1G01010   0   0    0   0   0   0    64    18     0     0    87   381    13
AT1G01020   2   0    7 708   0   0   422   602   125    58   181   148   122
AT1G01030   0   0    0   0   0   0     0     0     0     0     0     0     0
AT1G01040   0   0    0 459 533  42   205     2   577   201     0     0   112
AT1G01046   0   0    0   0   0   0     0     0     0     0     0     0     0
AT1G01050   2 166 1245   3  36   2  3865  5126  3804  1464  3072  3760  2112
```

Load the count table (new version with all seven GL2 cells), then subset to the samples from the GL2 cells (with "pGL" in the column name), and simplify the column names.

2 Analysis of the GL2 cell data

2.1 Preparing the count table

We subset the count table to only the columns referring to GL2 cells and clean up the column names.

```
countsAll <- fullCountTable[, substr( colnames(fullCountTable), 1, 3 ) == "GL2" ]
colnames(countsAll) <- gsub( "\\.", "-", colnames(countsAll) )
```

We split the count table in three sub-tables, one for the A. thalina plant genes ("At"), one for the HeLa genes ("HeLa") and one for the IVT spikes ("Sp"). The first two letter of the gene IDs (row names) are used for this categorization.

```
geneTypes <- factor( c( AT="At", pG="pGIBS", EN="HeLa", ER="ERCC" )[
   substr( rownames(countsAll), 1, 2 ) ] )
countsHeLa <- countsAll[ which( geneTypes=="HeLa" ), ]
countsAt <- countsAll[ which( geneTypes=="At" ), ]
countsSp <- countsAll[ which( geneTypes %in% c( "pGIBS", "ERCC" ) ), ]
```

We will not use the IVT spike data in this analysis. The other two tables now look as follows.

```
head( countsHeLa )
```

```
                GL2-1 GL2-2 GL2-3 GL2-4 GL2-5 GL2-6 GL2-7
ENSG00000000003   581  1850  2169   392  2046  1225   166
ENSG00000000005     0     0     0     0     0     0     0
ENSG00000000419   393  1263  1411   296  1754   247   159
ENSG00000000457     1   109   118    71    52     0     1
ENSG00000000460    89   179   419    42   310    16   116
ENSG00000000938     0     0     0     0     0     0     0
```

```
head( countsAt )
```

```
          GL2-1 GL2-2 GL2-3 GL2-4 GL2-5 GL2-6 GL2-7
AT1G01010    64    18     0     0    87   381    13
AT1G01020   422   602   125    58   181   148   122
AT1G01030     0     0     0     0     0     0     0
AT1G01040   205     2   577   201     0     0   112
AT1G01046     0     0     0     0     0     0     0
AT1G01050  3865  5126  3804  1464  3072  3760  2112
```

For later use, we get a translation of gene IDs to gene symbols.

```
geneSymbolsAt <- rownames(countsAt)
names( geneSymbolsAt ) <- rownames(countsAt)
hasSymbol <- rownames(countsAt) %in% Lkeys( org.At.tairSYMBOL )
symtbl <- toTable( org.At.tairSYMBOL[ rownames(countsAt)[ hasSymbol ] ] )
symtbl <- symtbl[ !duplicated( symtbl$gene_id ), ]
geneSymbolsAt[ symtbl$gene_id ] <- symtbl$symbol

head( geneSymbolsAt )
```

```
  AT1G01010   AT1G01020   AT1G01030   AT1G01040   AT1G01046   AT1G01050
  "ANAC001"      "ARV1"      "NGA3"      "ASU1" "AT1G01046"    "AtPPa1"
```

2.2 Normalization

We use the function "estimateSizeFactorsForMatrix" from DESeq to get size factors. This function calculates size factors as described in the DESeq paper (Anders and Huber, 2010) and in the Online Methods of the present paper.

```
sfHeLa <- estimateSizeFactorsForMatrix( countsHeLa )
sfAt <- estimateSizeFactorsForMatrix( countsAt)
```

See the size factors and their ratios:

```
rbind( HeLa = sfHeLa, At = sfAt, ratio = sfAt / sfHeLa )
```

```
          GL2-1    GL2-2     GL2-3     GL2-4     GL2-5     GL2-6     GL2-7
HeLa  0.5173383 2.022511 2.8804054 0.8552699 2.0415875 0.9551039 0.3109511
At    1.0234077 1.933931 1.7603544 1.3696320 0.8933595 0.8634580 0.5341586
ratio 1.9782176 0.956203 0.6111481 1.6014033 0.4375808 0.9040461 1.7178219
```

Divide by the size factors to get normalized counts. (Note the double use of "t" to make sure that columns, not rows, are divided by the size factors.)

```
nCountsHeLa <- t( t(countsHeLa) / sfHeLa )
nCountsAt <- t( t(countsAt) / sfAt )
```

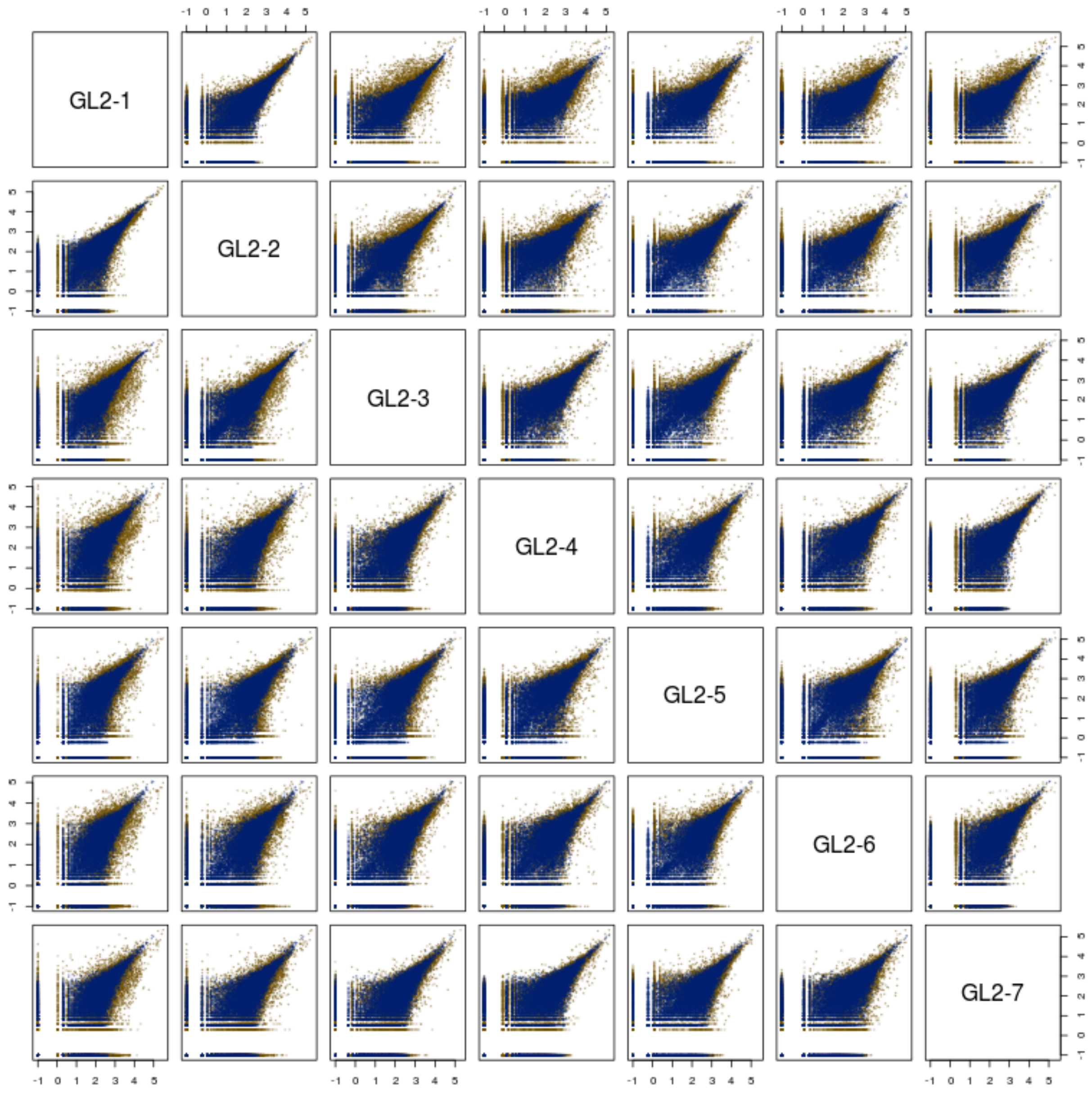
2.3 Plot of normalized counts

We use the following colours in our plots:

```
colHeLa <- "#00207040"
colAt <- "#70500040"
colAtHi <- "#B0901040"
```

Make a pairs plot of all GL-2 cells (Supplementary Figure 5).

```
pairs( log10( .1 + rbind( nCountsAt, nCountsHeLa ) ), pch=19, cex=.2,
   col = c( rep( colAt, nrow(nCountsAt) ), rep( colHeLa, nrow(nCountsHeLa) ) ) )
```



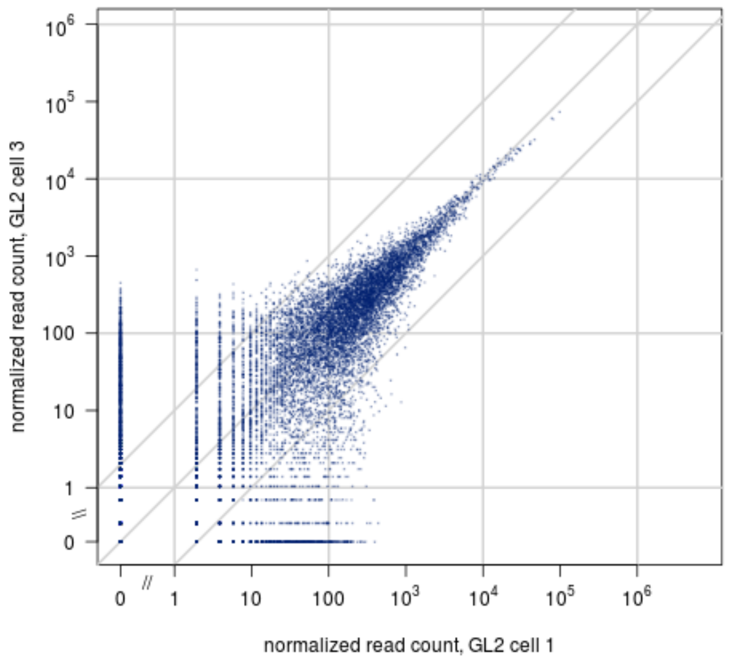
As the pairs plot is a bit large, we plot one comparsion a bit bigger, namely cells 1 vs 3 (Figures 2a and 2b).

First, the following function makes nice axes etc. This function (and most of the code for plotting in the following sections) is somewhat specialized to our data. users wishing to follow this code to perform analyses of their own data may want to use simpler, more standard, R code for plotting.

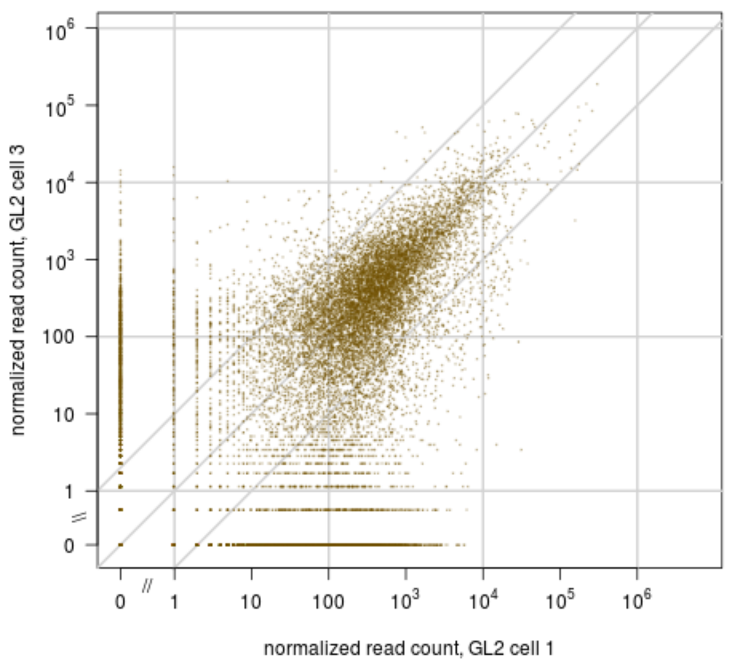
```
geneScatterplot <- function( x, y, xlab, ylab, col ) {
   plot( NULL, xlim=c( -.1, 6.2 ), ylim=c( -1, 6.2 ),
      xaxt="n", yaxt="n", xaxs="i", yaxs="i", asp=1,
      xlab=xlab, ylab=ylab )
   abline( a=-1, b=1, col = "lightgray", lwd=2 )
   abline( a=0, b=1, col = "lightgray", lwd=2 )
   abline( a=1, b=1, col = "lightgray", lwd=2 )
   abline( h=c(0,2,4,6), v=c(0,2,4,6), col = "lightgray", lwd=2 )
   points(
     ifelse( x > 0, log10(x), -.7 ),
     ifelse( y > 0, log10(y), -.7 ),
     pch=19, cex=.2, col = col )
   axis( 1, c( -.7, 0:6 ),
      c( "0", "1", "10", "100", expression(10^3), expression(10^4),
         expression(10^5), expression(10^6) ) )
   axis( 2, c( -.7, 0:6 ),
      c( "0", "1", "10", "100", expression(10^3), expression(10^4),
      expression(10^5), expression(10^6) ), las=2 )
   axis( 1, -.35, "//", tick=FALSE, line=-.7 )
   axis( 2, -.35, "\\\\", tick=FALSE, line=-.7 )
}
```

Now make the two plots, first Figure 2a, then 2b:

```
geneScatterplot( nCountsHeLa[,1], nCountsHeLa[,3],
   "normalized read count, GL2 cell 1", "normalized read count, GL2 cell 3",
   colHeLa )
```



```
geneScatterplot( nCountsAt[,1], nCountsAt[,3],
   "normalized read count, GL2 cell 1", "normalized read count, GL2 cell 3",
   colAt )
```



2.4 Estimating technical noise

We start by estimating the sample moments per gene.

```
meansHeLa <- rowMeans( nCountsHeLa )
varsHeLa <- rowVars( nCountsHeLa )
cv2HeLa <- varsHeLa / meansHeLa^2
```

Next, we define a minimum mean value to exclude genes with low mean and hence high CV from fit as they would otherwise skew it downwards (Online Methods and Supplementary Notes 6 and 7).

```
minMeanForFit <- unname( quantile( meansHeLa[ which( cv2HeLa > .3 ) ], .95 ) )
minMeanForFit
```

```
[1] 417.6733
```

Perform the fit of technical noise strength on average count. We regress cv2HeLa on 1/meansForHeLa. We use the glmgam.fit function from the statmod package to perform the regression as a GLM fit of the gamma family with log link. The 'cbind' construct serves to produce a model matrix with an intercept.

```
useForFit <- meansHeLa >= minMeanForFit
fit <- glmgam.fit( cbind( a0 = 1, a1tilde = 1/meansHeLa[useForFit] ),
   cv2HeLa[useForFit] )
fit$coefficients
```

```
          a0      a1tilde
  0.04245659 179.66783886
```

To get the actual noise coefficients, we need to subtract Xi (see Supplementary Note 6 for the difference between a1tilde and a1).

```
xi <- mean( 1 / sfHeLa )
```

```
a0 <- unname( fit$coefficients["a0"] )
a1 <- unname( fit$coefficients["a1tilde"] - xi )

c( a0, a1 )
```

```
[1]   0.04245659 178.42547279
```

2.4.1 Plot of the fit

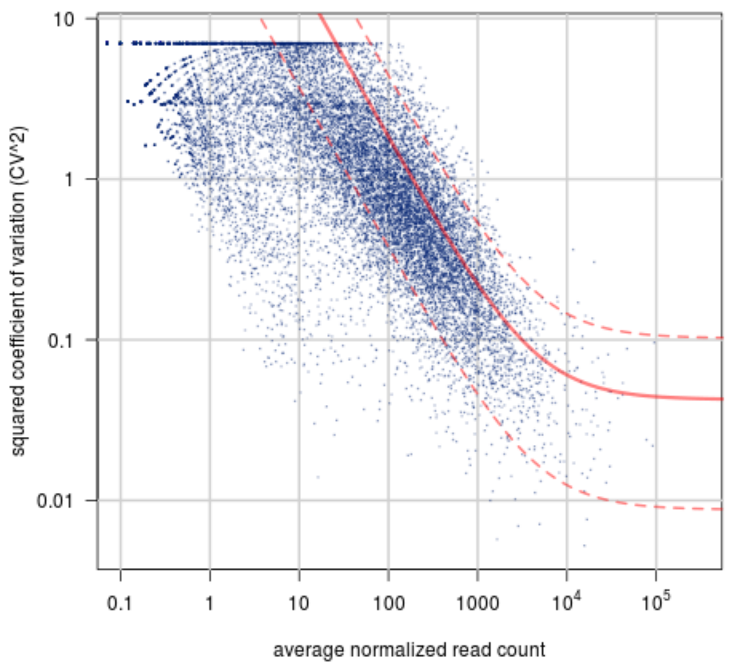
The following code produces the plot of the fit shown in Figure 2c.

```
# Prepare the plot (scales, grid, labels, etc.)
plot( NULL, xaxt="n", yaxt="n",
   log="xy", xlim = c( 1e-1, 3e5 ), ylim = c( .005, 8 ),
   xlab = "average normalized read count", ylab = "squared coefficient of variation (CV^2)"
axis( 1, 10^(-1:5), c( "0.1", "1", "10", "100", "1000",
  expression(10^4), expression(10^5) ) )
axis( 2, 10^(-2:1), c( "0.01", "0.1", "1", "10" ), las=2 )
abline( h=10^(-2:1), v=10^(-1:5), col="#D0D0D0", lwd=2 )

# Add the data points
points( meansHeLa, cv2HeLa, pch=20, cex=.2, col=colHeLa )

# Plot the fitted curve
xg <- 10^seq( -2, 6, length.out=1000 )
lines( xg, (xi+a1)/xg + a0, col="#FF000080", lwd=3 )

# Plot quantile lines around the fit
df <- ncol(countsAt) - 1
lines( xg, ( (xi+a1)/xg + a0  ) * qchisq( .975, df ) / df,
   col="#FF000080", lwd=2, lty="dashed" )
lines( xg, ( (xi+a1)/xg + a0  ) * qchisq( .025, df ) / df,
      col="#FF000080", lwd=2, lty="dashed" )
```



2.5 Testing plant genes for high variance

To perform the actual test (Online Methods, Supplementary Note 6), we start with again calculating the sample moments, now for the plant genes.

```
meansAt <- rowMeans( nCountsAt )
varsAt <- rowVars( nCountsAt )
cv2At <- varsAt / meansAt^2
```

The following is the term Psi + a0 * Theta, that appears in the formula for Omega (see formula in Online Methods).

```
psia1theta <- mean( 1 / sfAt ) + a1 * mean( sfHeLa / sfAt )
```

Now, we perform a one-sided test against the null hypothesis that the true variance is at most the technical variation plus biological variation with a CV of at most 50% (minBiolDisp = $.5^2$).

```
minBiolDisp <- .5^2
```

Calculate Omega, then perform the test, using the formula given in the Online methods and in Supplementary Note 6.

```
m <- ncol(countsAt)
cv2th <- a0 + minBiolDisp + a0 * minBiolDisp
testDenom <- ( meansAt * psia1theta + meansAt^2 * cv2th ) / ( 1 + cv2th/m )

p <- 1 - pchisq( varsAt * (m-1) / testDenom, m-1 )
```

Adjust for multiple testing with the Benjamini-Hochberg method, cut at 10%:

```
padj <- p.adjust( p, "BH" )
sig <- padj < .1
sig[is.na(sig)] <- FALSE

table( sig )
```

```
sig
FALSE  TRUE
32726   876
```

2.5.1 Plot

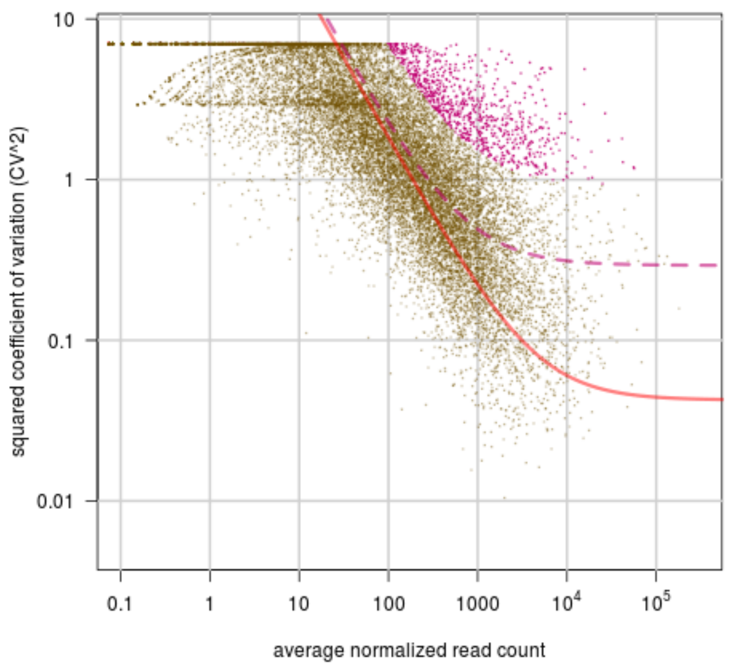
Make a plot of the plant genes, highlighting the highly variable ones (Figure 2d).

```
# Prepare plot in the same manner as before
plot( NULL, xaxt="n", yaxt="n",
   log="xy", xlim = c( 1e-1, 3e5 ), ylim = c( .005, 8 ),
   xlab = "average normalized read count", ylab = "squared coefficient of variation (CV^2)"
axis( 1, 10^(-1:5), c( "0.1", "1", "10", "100", "1000",
  expression(10^4), expression(10^5) ) )
axis( 2, 10^(-2:1), c( "0.01", "0.1", "1", "10" ), las=2 )
abline( h=10^(-2:1), v=10^(-1:5), col="#D0D0D0", lwd=2 )

# Plot the plant genes, use a different color if they are highly variable
points( meansAt, cv2At, pch=20, cex=.2,
   col = ifelse( padj < .1, "#C0007090", colAt ) )

# Add the technical noise fit, as before
xg <- 10^seq( -2, 6, length.out=1000 )
lines( xg, (xi+a1)/xg + a0, col="#FF000080", lwd=3 )
```

```
# Add a curve showing the expectation for the chosen biological CV^2 thershold
lines( xg, psia1theta/xg + a0 + minBiolDisp, lty="dashed", col="#C0007090", lwd=3 )
```



2.5.2 Table of highly variable genes

Most highly variable genes are strong only in one or two cells. To show this, we calculate the log ratio of the genes' expression in each cell to the mean:

```
log2RelExprAt <- log2( nCountsAt / meansAt )
```

We use this to produce a table of all significantly highly variable genes with some extra information, namely the mean normalized count, the log2 fold change of each cell to this mean, and the information which cell has the strongest expression.

```
highVarTable <- data.frame(
   row.names = NULL,
   geneID = rownames(countsAt)[ sig ],
   geneSymbol = geneSymbolsAt[ sig ],
   meanNormCount = meansAt[ sig ],
   strongest = factor( colnames( log2RelExprAt )[
      apply( log2RelExprAt[ sig, ], 1, which.max ) ] ),
   log2RelExprAt[ sig, ],
   check.names=FALSE )

head( highVarTable )
```

```
     geneID geneSymbol meanNormCount strongest      GL2-1      GL2-2     GL2-3      GL2-4
1 AT1G01100  AT1G01100     2534.5561     GL2-1  1.6221783  1.0724231 -2.119163 -2.3754433
2 AT1G01140      CIPK9     1584.2303     GL2-6       -Inf       -Inf -1.293147 -0.8690356
3 AT1G01730  AT1G01730      913.3887     GL2-3 -0.9406882  0.7364503  2.020811 -3.3940557
4 AT1G01740  AT1G01740      837.6666     GL2-3 -0.5463967  1.3201552  1.876914       -Inf
5 AT1G02230    ANAC004      208.7058     GL2-4 -0.5788364 -5.1974315 -4.273265  2.6442089
6 AT1G02780    emb2386    10119.6519     GL2-1  1.7777659  0.4076499 -2.532492 -1.8231270
       GL2-5      GL2-6      GL2-7
```

```
1  -0.4417918 -1.4518590 -1.7662327
2 -10.4668792  2.5764124 -3.6588172
3  -3.4629446 -9.6232830 -0.8166829
4        -Inf -4.7979905 -3.1908625
5        -Inf       -Inf       -Inf
6  -0.4174578 -0.7624331 -1.1493215
```

We write out the table. It is given in the Supplement as Supplementary Table 2.

```
write.csv( highVarTable, file="highly_variant_genes_GL2.tsv", row.names=FALSE )
```

2.6 GO analysis

Next, we check whether the high-variance genes are enriched in certain GO categories using TopGO (Online Methods).

To work only with genes with uniform power to detect high or low variance, we include only genes with an average count above 300.

```
minCountForEnrichment <- 300
```

The following function performs the analysis. It takes a vector with gene IDs (here: all plant genes) and then two Boolean vectors of the same length, the first indicating which genes are to be included in the analysis (here: above the mean cut-off) and the second indicating which genes are significant (i.e., highly variable).

```
topGOAnalysis <- function( geneIDs, inUniverse, inSelection )
   sapply( c( "MF", "BP", "CC" ), function( ont ) {
      alg <- factor( as.integer( inSelection[inUniverse] ) )
      names(alg) <- geneIDs[inUniverse]
      tgd <- new( "topGOdata", ontology=ont, allGenes = alg, nodeSize=5,
         annot=annFUN.org, mapping="org.At.tair.db" )
      resultTopGO <- runTest(tgd, algorithm = "elim", statistic = "Fisher" )
      GenTable( tgd, resultTopGO, topNodes=15 ) },
   simplify=FALSE )
```

We use this to perform an enrichment analysis for the list of highly variable genes:

```
goResults <-
  topGOAnalysis(
     rownames(countsAt),
     meansAt >= minCountForEnrichment & !is.na(padj),
     padj < .1 )
```

2.6.1 Results:

How to correctly adjust for multiple testing in enrichment analyses is slightly controversal, so to keep it simple, we consider categories with raw p value below 10^{-5} as clearly and below 10^{-4} as maybe significant. Categories thus deemed significant have been taken from the results below and are listed in Supplemenary Table 4.

```
goResults[["MF"]]
```

```
        GO.ID                                        Term Annotated Significant Expected re
1  GO:0003735         structural constituent of ribosome       212         140    23.24 < 
2  GO:0015250                        water channel activity        17           9     1.86 2.
3  GO:0003677                                   DNA binding       391          69    42.86 2.
4  GO:0003746      translation elongation factor activity        18           7     1.97  0
5  GO:0019843                                  rRNA binding         7           4     0.77  0
6  GO:0016762 xyloglucan:xyloglucosyl transferase acti...         8           4     0.88  0
7  GO:0016884 carbon-nitrogen ligase activity, with gl...         8           4     0.88  0
8  GO:0003676                          nucleic acid binding       706         123    77.39  0
```

```
9  GO:0003723                                  RNA binding       229           44    25.10  0
10 GO:0016847 1-aminocyclopropane-1-carboxylate syntha...         6            3     0.66  0
11 GO:0004553 hydrolase activity, hydrolyzing O-glycos...        86           16     9.43  0
12 GO:0016798 hydrolase activity, acting on glycosyl b...        94           17    10.30  0
13 GO:0016684 oxidoreductase activity, acting on perox...        28            7     3.07  0
14 GO:0004601                           peroxidase activity        28            7     3.07  0
15 GO:0003690                     double-stranded DNA binding         8            3     0.88  0
```

```
goResults[["CC"]]
```

```
        GO.ID                                         Term Annotated Significant Expected re
1  GO:0022625           cytosolic large ribosomal subunit        92          78    10.04 <
2  GO:0022627           cytosolic small ribosomal subunit        69          48     7.53 <
3  GO:0005730                                   nucleolus       161          71    17.56 5.
4  GO:0000786                                  nucleosome        41          33     4.47 3.
5  GO:0009506                                 plasmodesma       478         108    52.15 5.
6  GO:0005618                                   cell wall       255          52    27.82 3.
7  GO:0022626                          cytosolic ribosome       207         139    22.58 9.
8  GO:0005576                        extracellular region       126          24    13.75  0
9  GO:0005774                           vacuolar membrane       334          51    36.44  0
10 GO:0005634                                     nucleus       919         165   100.26  0
11 GO:0009507                                 chloroplast       745         100    81.28  0
12 GO:0005773                                     vacuole       532          82    58.04  0
13 GO:0009536                                     plastid       784         102    85.53  0
14 GO:0048046                                    apoplast       102          17    11.13  0
15 GO:0005853 eukaryotic translation elongation factor...         5           2     0.55  0
```

```
goResults[["BP"]]
```

```
        GO.ID                                         Term Annotated Significant Expected re
1  GO:0001510                             RNA methylation       105          79    11.42 <
2  GO:0006412                                 translation       448         155    48.74 <
3  GO:0042254                         ribosome biogenesis       108          63    11.75 7.
4  GO:0006334                         nucleosome assembly        45          34     4.90 2.
5  GO:0009220 pyrimidine ribonucleotide biosynthetic p...        53          29     5.77 3.
6  GO:0006364                             rRNA processing        39          19     4.24 4.
7  GO:0006414                      translational elongation        33          14     3.59 3.
8  GO:0042545                       cell wall modification        79          22     8.59 2.
9  GO:0009664                  plant-type cell wall organization       106          26    11.53 4.
10 GO:0000724 double-strand break repair via homologou...         9           6     0.98 0.
11 GO:0006164         purine nucleotide biosynthetic process        50          15     5.44 0.
12 GO:1901070 guanosine-containing compound biosynthet...         5           4     0.54 0.
13 GO:0009646                   response to absence of light        20           8     2.18 0.
14 GO:0009955          adaxial/abaxial pattern specification        12           6     1.31 0.
15 GO:0000462 maturation of SSU-rRNA from tricistronic...         6           4     0.65 0.
```

2.7 Heatmap

This section describes how Supplementary Figure 9a was produced.

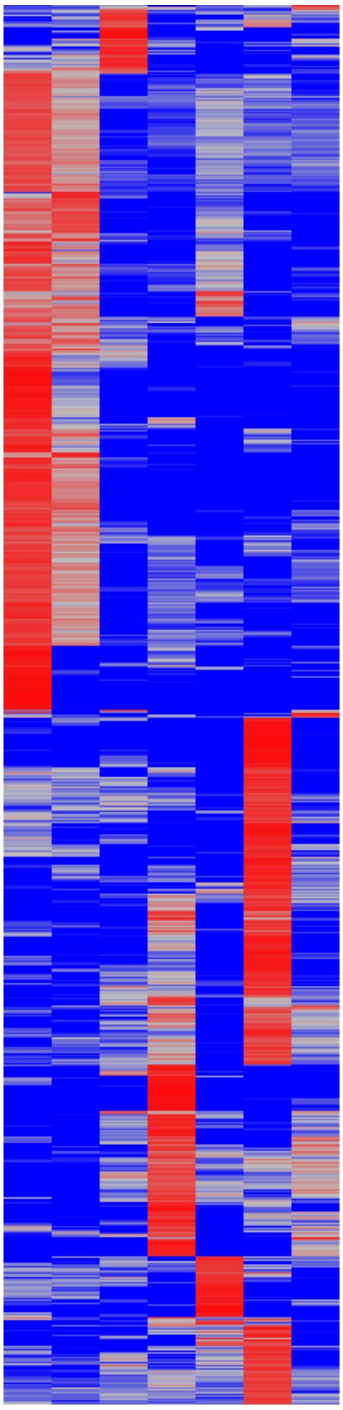
In order to get the genes aligned to the plot's pixels, we use the following custom heatmap function instead of R's standard one. It expects a matrix with values in the range given by 'zlim' (everything outside this range will be pulled in to its margins), a number pair for zlim, a color palette, and two integers for the width and height in pixels that should be used to represent each matrix element.

```
pixelHeatmap <- function( m, zlim=range(m), col=colorRampPalette(c("blue","orange"))(100),
      pxWidth=1, pxHeight=1 ) {
   col <- col2rgb(col)/255
   mn <- ( m - zlim[1] ) / ( zlim[2] - zlim[1] )
   mn[ mn<0 ] <- 0
   mn[ mn>1 ] <- 1
   mn <- 1 + round( mn * (ncol(col)-1) )
   a <- array( NA_real_, c( nrow(m)*pxWidth, ncol(m)*pxHeight, 3 ) )
   for( i in 1:nrow(mn) )
      for( j in 1:ncol(mn) )
         a[ (((i-1)*pxWidth)+1) : (i*pxWidth), (((j-1)*pxHeight)+1) : (j*pxHeight),  ] <-
            rep( col[ , mn[ i, j ] ] , each = pxWidth*pxHeight )
   Image( a, colormode="color" )
}
```

We use this to plot the relative expressions of the highly vaiable genes.

```
relSig <- log2RelExprAt[ sig, ]
relSig[ relSig < -4 ] <- -4
ord <- hclust(dist(relSig))$order
hmSig <- pixelHeatmap( t( relSig[ord,] ), zlim=c( -3, 3 ), pxWidth=30,
   col=colorRampPalette(c("blue","gray","red"))(100) )
```

```
writeImage( hmSig, "variableGenes_heatmap.png" )
```



We annotate this heatmap with a few hand-picked GO terms:

```
someGOTerms <- c( "GO:0003735", "GO:0001510", "GO:0005730", "GO:0009506",
   "GO:0003677", "GO:0000786", "GO:0012505", "GO:0009269" )

unname( t( sapply( someGOTerms, function(x) toTable(GOTERM[ x ])[1,2:3] ) ) )
```

```
     [,1]         [,2]
[1,] "GO:0003735" "structural constituent of ribosome"
[2,] "GO:0001510" "RNA methylation"
[3,] "GO:0005730" "nucleolus"
[4,] "GO:0009506" "plasmodesma"
[5,] "GO:0003677" "DNA binding"
[6,] "GO:0000786" "nucleosome"
```

```
[7,] "GO:0012505" "endomembrane system"
[8,] "GO:0009269" "response to desiccation"
```

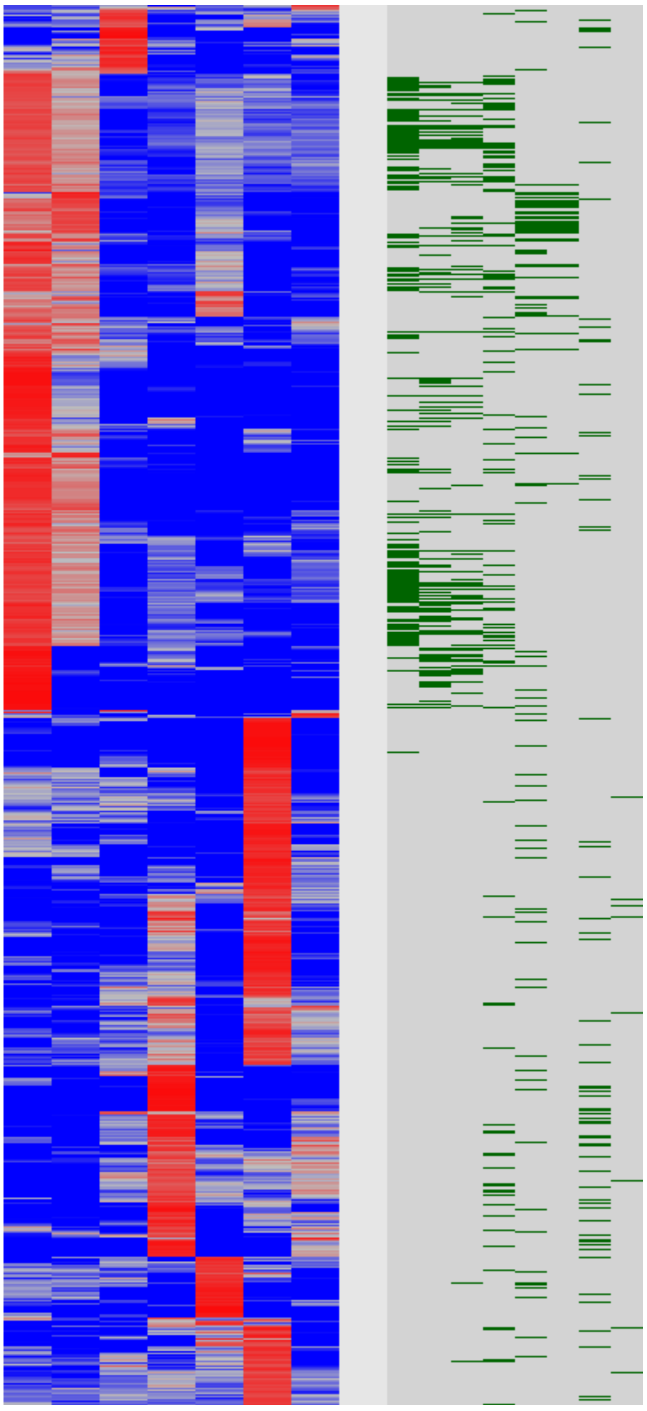
For these, make a second heatmap indicating the genes' membership.

```
sigInGO <- sapply( unname(someGOTerms), function(go)
   rownames(countsAt)[sig] %in% org.At.tairGO2TAIR[[ go ]] )

hmGO <- pixelHeatmap( t(sigInGO[ ord, ]), col=c( "lightgray", "darkgreen" ), pxWidth=20 )
```

Put the two heatmaps next to each other.

```
spacer <- Image( array( .9, 30 * 3 * sum(sig), dim=c( 30, sum(sig), 3 ) ), colormode="color
writeImage(
   Image( abind::abind( hmSig, spacer, hmGO, along=1 ), colormode="color" ),
   files="heatmap.png" )
```



2.8 Read downsampling

To show that we could have worked with fewer reads (as claimed in Supplementary Note 9), we downsample all read counts to 1/5 using draws from binomial distributions

```
dsfrac <- 0.2

countsAt_ds <- apply( countsAt, 1:2, function(k) rbinom( 1, k, dsfrac ) )
countsHeLa_ds <- apply( countsHeLa, 1:2, function(k) rbinom( 1, k, dsfrac ) )
```

We redo the analysis as before.

First, the normalization:

```
sfAt_ds <- estimateSizeFactorsForMatrix( countsAt_ds )
sfHeLa_ds <- estimateSizeFactorsForMatrix( countsHeLa_ds )

nCountsAt_ds <- t( t(countsAt_ds) / sfAt_ds )
nCountsHeLa_ds <- t( t(countsHeLa_ds) / sfHeLa_ds )
```

Next, the technical noise fit:

```
meansHeLa_ds <- rowMeans( nCountsHeLa_ds )
varsHeLa_ds <- rowVars( nCountsHeLa_ds )
cv2HeLa_ds <- varsHeLa_ds / meansHeLa_ds^2

minMeanForFit_ds <- unname( quantile( meansHeLa_ds[ which( cv2HeLa_ds > .3 ) ], .95 ) )
useForFit_ds <- meansHeLa_ds >= minMeanForFit_ds
fit_ds <- glmgam.fit( cbind( a0 = 1, a1tilde = 1/meansHeLa_ds[useForFit_ds] ),
   cv2HeLa_ds[useForFit_ds] )

xi_ds <- mean( 1 / sfHeLa_ds )
a0_ds <- unname( fit_ds$coefficients["a0"] )
a1_ds <- unname( fit_ds$coefficients["a1tilde"] - xi_ds )

c( a0_ds, a1_ds )
```

```
[1]  0.03706779 35.43892577
```

Note how this a0 stayed roughly the same, and a1 got reduced according to the downsampling fraction; compare with

```
c( a0, a1 * dsfrac )
```

```
[1]  0.04245659 35.68509456
```

Finally, the test for high variability.

```
meansAt_ds <- rowMeans( nCountsAt_ds )
varsAt_ds <- rowVars( nCountsAt_ds )

psia1theta_ds <- mean( 1 / sfAt_ds ) + a1_ds * mean( sfHeLa_ds / sfAt_ds )
cv2th_ds <- a0_ds + minBiolDisp + a0_ds * minBiolDisp
testDenom_ds <- ( meansAt_ds * psia1theta_ds + meansAt_ds^2 * cv2th_ds ) / ( 1 + cv2th_ds/m

p_ds <- 1 - pchisq( varsAt_ds * (m-1) / testDenom_ds, m-1 )
padj_ds <- p.adjust( p_ds, "BH" )
```

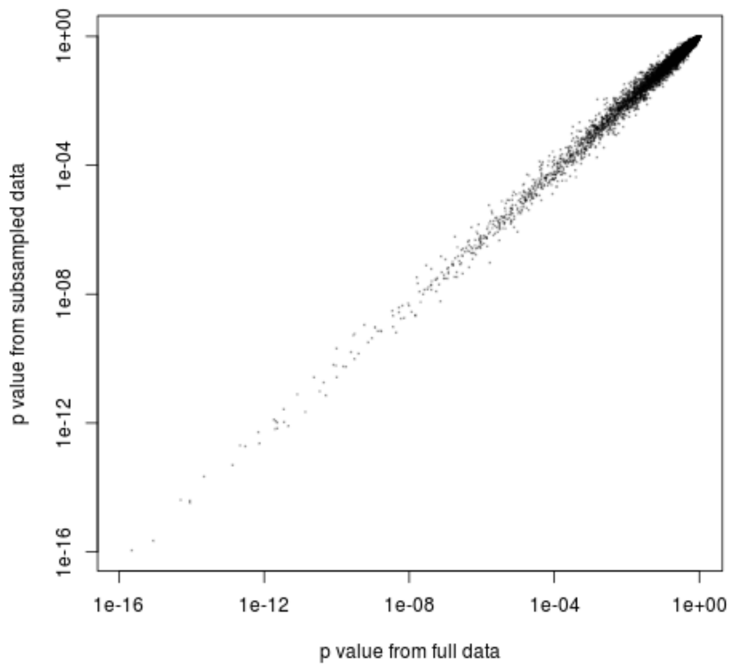
Compare the results and calculate the overlap:

```
addmargins( table( all_reads = padj < .1, subsampled = padj_ds < .1 ) )
```

```
         subsampled
all_reads FALSE  TRUE   Sum
    FALSE 17313    64 17377
    TRUE     27   849   876
    Sum   17340   913 18253
```

Also, a plot of the p values (Supplementary Figure 12; note that this plots always changes slightly due to the random nature of the downsampling)

```
plot( p, p_ds,
   log="xy", pch=19, cex=.2, col="#00000040",
   xlab = "p value from full data", ylab="p value from subsampled data" )
```



2.9 Effect of transcript length

Here, we test whether it is benefitial to normalize counts for transcript length.

We load a table of transcript lengths for the human genome (given in Supplementary Table 9):

```
a <- read.csv( "Supplementary_Table_9.csv" )
humanGeneLengths <- a$length
names(humanGeneLengths) <- a$geneID
head(humanGeneLengths)
```

```
ENSG00000000003 ENSG00000000005 ENSG00000000419 ENSG00000000457 ENSG00000000460 ENSG0000000
           2968            1610            1207            3844            6354
```

We use this to convert the normalized counts to normalized counts per kilobase transcript length.

```
nCountsHeLaPK <- nCountsHeLa / humanGeneLengths[rownames(nCountsHeLa)] * 1000
```

We redo the fit using this data (removing a single gene for which we are missing length information):

```
useForFitPK <- useForFit
useForFitPK["ENSG00000242125"] <- FALSE
meansHeLaPK <- rowMeans( nCountsHeLaPK )
fitPK <- glmgam.fit( cbind( a0 = 1, a1tilde = 1/meansHeLaPK[useForFitPK] ),
   cv2HeLa[useForFitPK] )
```

We compare the residual variances from both fits:

```
totalVariance <- var( log( cv2HeLa[useForFit] ) )
residualVariance <- var( log( cv2HeLa[useForFit] ) - log( fitted.values(fit) ) )

totalVariancePK <- var( log( cv2HeLa[useForFitPK] ) )
residualVariancePK <- var( log( cv2HeLa[useForFitPK] ) - log( fitted.values(fitPK) ) )
```

Here are the variances and their ratios:

```
c( residualVariance, totalVariance, residualVariance / totalVariance )
c( residualVariancePK, totalVariancePK, residualVariancePK / totalVariancePK )
```

```
[1] 0.5438976 0.8333216 0.6526863
[1] 0.6610947 0.8334789 0.7931751
```

Here, the length-normalization increases the residual variance, i.e., is not benefitial.

Note: These residual variance fractions may seem large. However, bear in mind that even a perfect fit cannot explain the sampling variance of the log CV^2 estimates, which approximately have variance 2/(m-1), i.e. account for the following fraction of the total variance:

```
2/(m-1) / totalVariance
```

```
[1] 0.4000057
```

This is more than half of the residual variance fraction. Hence, if we subtract this from the tota variance to get the total "explainable" variance, we get a rather large fraction of explained variance:

```
explainedVariance <- totalVariance - residualVariance
explainableVariance <- totalVariance - 2/(m-1)

explainedVariance / explainableVariance
```

```
[1] 0.5788616
```

The same for the length-divided fit:

```
explainedVariancePK <- totalVariancePK - residualVariancePK
explainableVariancePK <- totalVariancePK - 2/(m-1)

explainedVariancePK / explainableVariancePK
```

```
[1] 0.344668
```

Finally, we save the current state, for convenience.

```
save.image( "GL2_analysis_image.RData" )
```

3 Analysis of the QC cells

The analysis of the QC cells is done in exactly the same way as for the GL2 cells. Hence, we give here the code, essentially the same code as before, without much comments.

As the code here re-uses the variable names used in the previous part, we clear the global environment in order to start with a clean slate.

```
rm( list=ls() )
```

3.1 Preparing the count table.

We read in the full count table, as before, and now subset to only the columns referring to QC cells.

```
fullCountTable <- read.csv( "Supplementary_Table_8.csv", header=TRUE, row.names=1 )
countsAll <- fullCountTable[, substr( colnames(fullCountTable), 1, 2 ) == "QC" ]
```

We split the count table again into three sub-tables

```
geneTypes <- factor( c( AT="At", pG="pGIBS", EN="HeLa", ER="ERCC" )[
   substr( rownames(countsAll), 1, 2 ) ] )
countsHeLa <- countsAll[ which( geneTypes=="HeLa" ), ]
countsAt <- countsAll[ which( geneTypes=="At" ), ]
countsSp <- countsAll[ which( geneTypes %in% c( "pGIBS", "ERCC" ) ), ]
```

Again, the gene symbols:

```
geneSymbolsAt <- rownames(countsAt)
names( geneSymbolsAt ) <- rownames(countsAt)
hasSymbol <- rownames(countsAt) %in% Lkeys( org.At.tairSYMBOL )
symtbl <- toTable( org.At.tairSYMBOL[ rownames(countsAt)[ hasSymbol ] ] )
symtbl <- symtbl[ !duplicated( symtbl$gene_id ), ]
geneSymbolsAt[ symtbl$gene_id ] <- symtbl$symbol

head( geneSymbolsAt )
```

```
  AT1G01010   AT1G01020   AT1G01030   AT1G01040   AT1G01046   AT1G01050
  "ANAC001"      "ARV1"      "NGA3"      "ASU1" "AT1G01046"    "AtPPa1"
```

3.2 Normalization

As before:

```
sfHeLa <- estimateSizeFactorsForMatrix( countsHeLa )
sfAt <- estimateSizeFactorsForMatrix( countsAt)
```

See the size factors and their ratios:

```
rbind( HeLa = sfHeLa, At = sfAt, ratio = sfAt / sfHeLa )
```

```
           QC1       QC2       QC3      QC4       QC5       QC6
HeLa  1.287133 0.8061698 0.8695366 1.206279 1.1365133 1.2384236
At    1.595883 1.2135275 1.2409527 2.315439 0.9654965 0.6946031
ratio 1.239875 1.5053001 1.4271426 1.919489 0.8495251 0.5608768
```

Divide by the size factors to get normalized counts

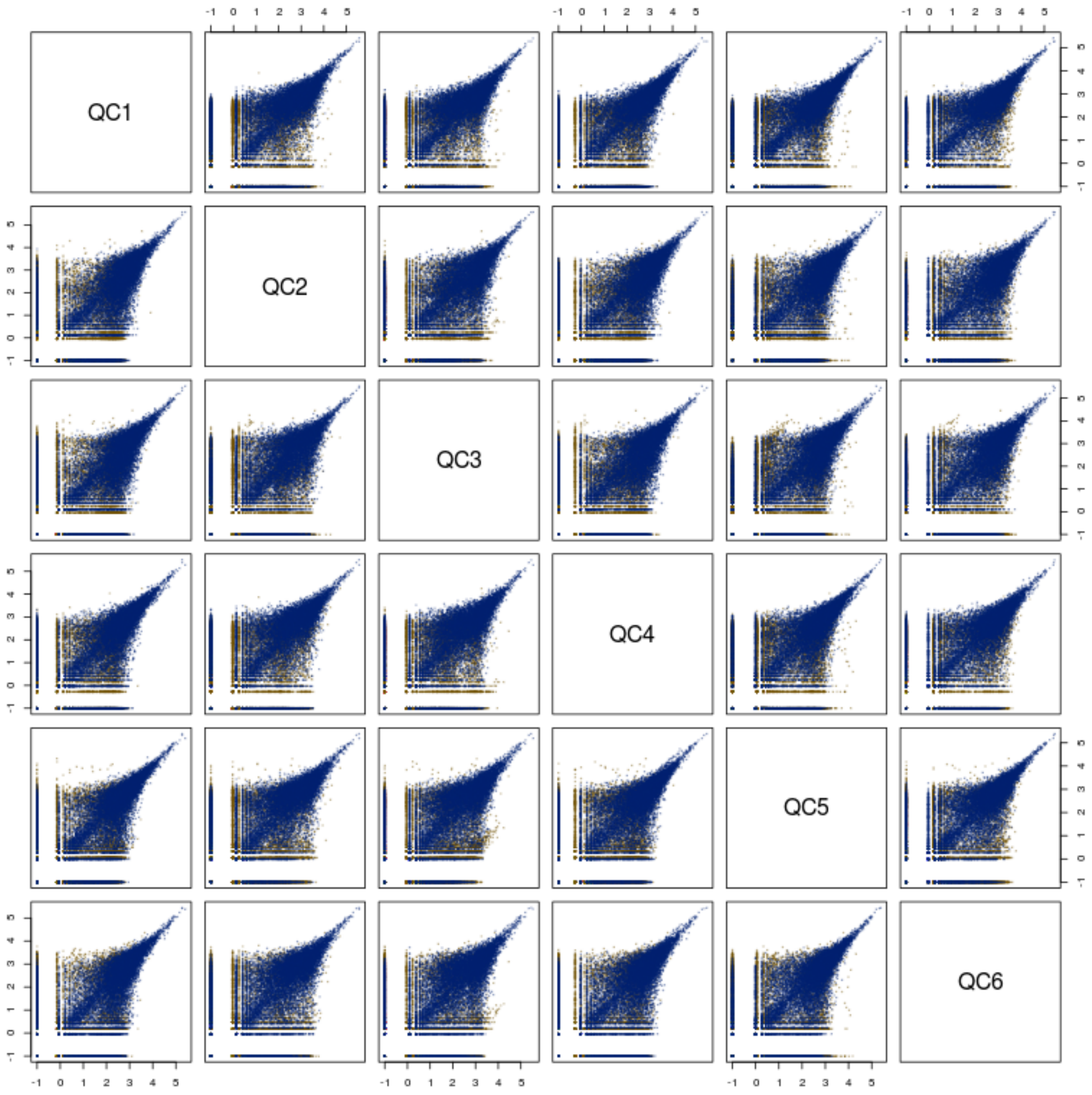
```
nCountsHeLa <- t( t(countsHeLa) / sfHeLa )
nCountsAt <- t( t(countsAt) / sfAt )
```

3.3 Plot of normalized counts

The plot shown in Supplementary Figure 6.

```
colHeLa <- "#00207040"
colAt <- "#70500040"
colAtHi <- "#B0901040"
```

```
pairs( log10( .1 + rbind( nCountsAt, nCountsHeLa ) ), pch=19, cex=.2,
   col = c( rep( colAt, nrow(nCountsAt) ), rep( colHeLa, nrow(nCountsHeLa) ) ) )
```



3.4 Estimating technical noise

Estimate sample moments per gene.

```
meansHeLa <- rowMeans( nCountsHeLa )
varsHeLa <- rowVars( nCountsHeLa )
cv2HeLa <- varsHeLa / meansHeLa^2
```

Find the minimum mean value for the fit.

```
minMeanForFit <- unname( quantile( meansHeLa[ which( cv2HeLa > .3 ) ], .95 ) )
minMeanForFit
```

```
[1] 1445.372
```

Perform the fit.

```
useForFit <- meansHeLa >= minMeanForFit
fit <- glmgam.fit( cbind( a0 = 1, a1tilde = 1/meansHeLa[useForFit] ),
   cv2HeLa[useForFit] )
fit$coefficients
```

```
          a0      a1tilde
  0.03848821 675.46699263
```

Subtract Xi.

```
xi <- mean( 1 / sfHeLa )

a0 <- unname( fit$coefficients["a0"] )
a1 <- unname( fit$coefficients["a1tilde"] - xi )

c( a0, a1 )
```

```
[1]   0.03848821 674.51970093
```

3.4.1 Plot of the fit

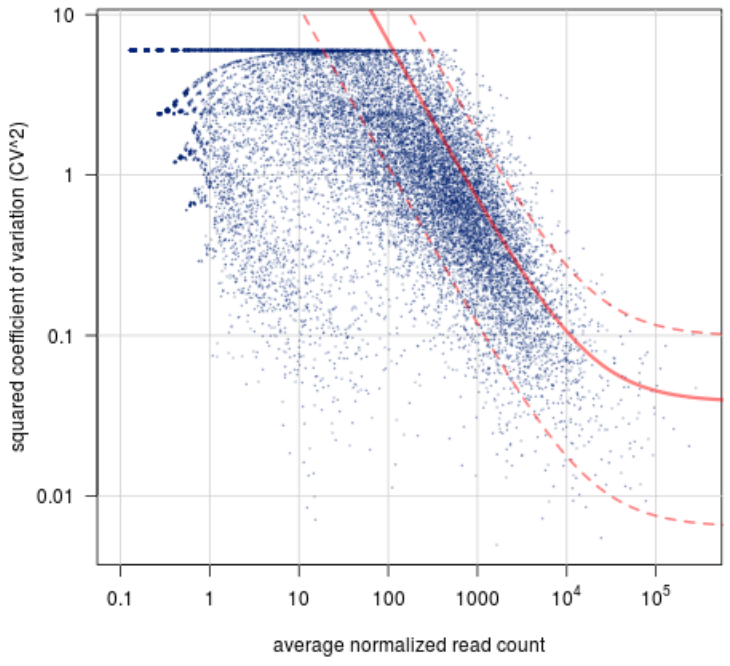
A plot of the fit, code as before (Supplementary Figure 8a).

```
# Prepare the plot (scales, grid, labels, etc.)
plot( NULL, xaxt="n", yaxt="n",
   log="xy", xlim = c( 1e-1, 3e5 ), ylim = c( .005, 8 ),
   xlab = "average normalized read count", ylab = "squared coefficient of variation (CV^2)"
axis( 1, 10^(-1:5), c( "0.1", "1", "10", "100", "1000",
  expression(10^4), expression(10^5) ) )
axis( 2, 10^(-2:1), c( "0.01", "0.1", "1", "10" ), las=2 )
abline( h=10^(-2:1), v=10^(-1:5), col="#D0D0D0" )

# Add the data points
points( meansHeLa, cv2HeLa, pch=20, cex=.2, col=colHeLa )

# Plot the fitted curve
xg <- 10^seq( -2, 6, length.out=1000 )
lines( xg, (xi+a1)/xg + a0, col="#FF000080", lwd=3 )


# Plot quantile lines around the fit
df <- ncol(countsAt) - 1
lines( xg, ( (xi+a1)/xg + a0  ) * qchisq( .975, df ) / df,
   col="#FF000080", lwd=2, lty="dashed" )
lines( xg, ( (xi+a1)/xg + a0  ) * qchisq( .025, df ) / df,
   col="#FF000080", lwd=2, lty="dashed" )
```



3.5 Testing plant genes for high variance

First the sample moments.

```
meansAt <- rowMeans( nCountsAt )
varsAt <- rowVars( nCountsAt )
cv2At <- varsAt / meansAt^2
```

Next, calculate Psi + a1 * Theta.

```
psia1theta <- mean( 1 / sfAt ) + a1 * mean( sfHeLa / sfAt )
```

We again test the null hypothesis that the biological CV is below 50%.

```
minBiolDisp <- .5^2
```

Calculate Omega, then perform the test

```
m <- ncol(countsAt)
cv2th <- a0 + minBiolDisp + a0 * minBiolDisp
testDenom <- ( meansAt * psia1theta + meansAt^2 * cv2th ) / ( 1 + cv2th/m )

p <- 1 - pchisq( varsAt * (m-1) / testDenom, m-1 )
```

Adjust for multiple testing, cut at 10%:

```
padj <- p.adjust( p, "BH" )
sig <- padj < .1
sig[is.na(sig)] <- FALSE

table( sig )
```

```
sig
```

```
FALSE  TRUE
33538    64
```

3.5.1 Plot

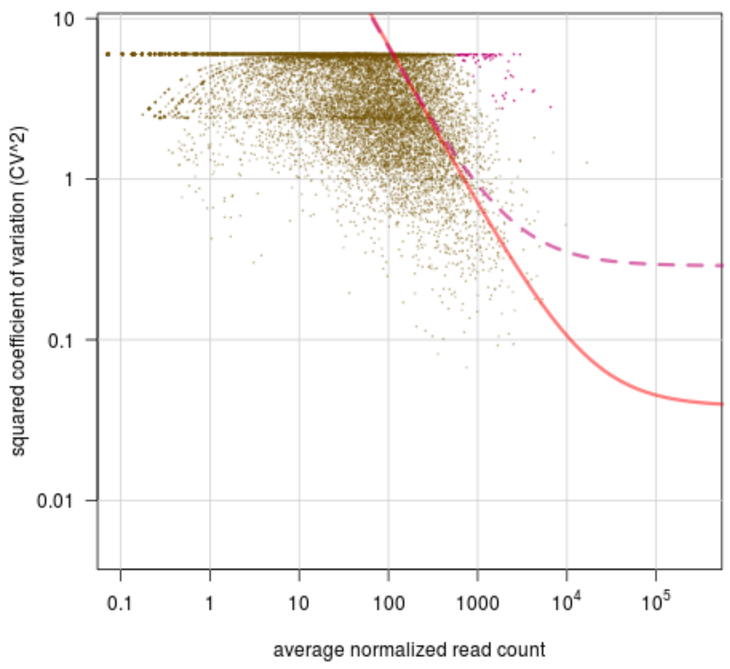
Same code as before, now to produce Supplementary Figure 8b.

```
# Prepare plot in the same manner as before
plot( NULL, xaxt="n", yaxt="n",
   log="xy", xlim = c( 1e-1, 3e5 ), ylim = c( .005, 8 ),
   xlab = "average normalized read count", ylab = "squared coefficient of variation (CV^2)" )
axis( 1, 10^(-1:5), c( "0.1", "1", "10", "100", "1000",
  expression(10^4), expression(10^5) ) )
axis( 2, 10^(-2:1), c( "0.01", "0.1", "1", "10" ), las=2 )
abline( h=10^(-2:1), v=10^(-1:5), col="#D0D0D0" )

# Plot the plant genes, use a different color if they are highly variable
points( meansAt, cv2At, pch=20, cex=.2,
   col = ifelse( padj < .1, "#C0007090", colAt ) )

# Add the technical noise fit, as before
xg <- 10^seq( -2, 6, length.out=1000 )
lines( xg, (xi+a1)/xg + a0, col="#FF000080", lwd=3 )

# Add a curve showing the expectation for the chosen biological CV^2 thershold
lines( xg, psia1theta/xg + a0 + minBiolDisp, lty="dashed", col="#C0007090", lwd=3 )
```



3.6 Table of highly variable genes

Write out Supplementary Table 3.

```
log2RelExprAt <- log2( nCountsAt / meansAt )

highVarTable <- data.frame(
   row.names = NULL,
   geneID = rownames(countsAt)[ sig ],
   geneSymbol = geneSymbolsAt[ sig ],
   meanNormCount = meansAt[ sig ],
   strongest = factor( colnames( log2RelExprAt )[
      apply( log2RelExprAt[ sig, ], 1, which.max ) ] ),
   log2RelExprAt[ sig, ],
   check.names=FALSE )

head( highVarTable )
```

```
     geneID geneSymbol meanNormCount strongest        QC1       QC2       QC3        QC4
1 AT1G02500     AtSAM1      1600.104       QC2  -1.488582  2.182097 -2.665379  -1.787801 -0
2 AT1G02690     IMPA-6      1478.025       QC3  -3.469101 -7.638737  2.549669  -5.054241 -6
3 AT1G02730    ATCSLD5      1842.709       QC3 -10.521968 -8.541857  2.577869 -10.473936 -5
4 AT1G03780     AtTPX2      1308.925       QC3  -5.706594 -4.545911  2.564255 -11.565453 -5
5 AT1G06760  AT1G06760      4292.894       QC5 -12.742090 -3.410304 -1.912597  -2.691243  2
6 AT1G07790       HTB1      1240.020       QC5  -3.458650 -9.555354      -Inf  -4.995580  2
         QC6
1 -7.7962823
2 -7.6817881
3 -7.3218736
4 -7.2434655
5 -0.6164412
6 -9.7504081
```

Write out the table

```
write.csv( highVarTable, file="highly_variant_genes_QC.tsv", row.names=FALSE )
```

3.7 GO analysis

The TopGo analysis, as before.

This time, we include only genes with an average count above 600.

```
minCountForEnrichment <- 600
```

The work function, as before.

```
topGOAnalysis <- function( geneIDs, inUniverse, inSelection )
   sapply( c( "MF", "BP", "CC" ), function( ont ) {
      alg <- factor( as.integer( inSelection[inUniverse] ) )
      names(alg) <- geneIDs[inUniverse]
      tgd <- new( "topGOdata", ontology=ont, allGenes = alg, nodeSize=5,
         annot=annFUN.org, mapping="org.At.tair.db" )
      resultTopGO <- runTest(tgd, algorithm = "elim", statistic = "Fisher" )
      GenTable( tgd, resultTopGO, topNodes=15 ) },
   simplify=FALSE )
```

The analysis.

```
goResults <-
  topGOAnalysis(
     rownames(countsAt),
     meansAt >= minCountForEnrichment & !is.na(padj),
     padj < .1 )
```

3.7.1 Results:

```
goResults[["MF"]]
```

```
        GO.ID                                        Term Annotated Significant Expected re
1  GO:0016538 cyclin-dependent protein kinase regulato...         5           5     0.59
2  GO:0003777                  microtubule motor activity         5           5     0.59
3  GO:0003677                                 DNA binding        65          19     7.66
4  GO:0019901                       protein kinase binding         7           5     0.83 0.
5  GO:0009055                    electron carrier activity         8           4     0.94 0.
6  GO:0046906                        tetrapyrrole binding        10           4     1.18 0.
7  GO:0020037                                heme binding        10           4     1.18 0.
8  GO:0005506                            iron ion binding        12           4     1.41 0.
9  GO:0016705 oxidoreductase activity, acting on paire...        10           3     1.18 0.
10 GO:0019825                              oxygen binding         6           2     0.71 0.
11 GO:0005200      structural constituent of cytoskeleton         6           2     0.71 0.
12 GO:0005488                                     binding       320          47    37.72 0.
13 GO:0019899                              enzyme binding        10           6     1.18 0.
14 GO:0016765 transferase activity, transferring alkyl...        10           2     1.18 0.
15 GO:0032559                 adenyl ribonucleotide binding        72          10     8.49 0.
```

```
goResults[["CC"]]
```

```
        GO.ID                                    Term Annotated Significant Expected resul
1  GO:0000786                              nucleosome        21          16     2.38 1.2e-
2  GO:0005874                             microtubule        12           5     1.36  0.00
3  GO:0005634                                 nucleus       152          24    17.24  0.02
4  GO:0005694                              chromosome        26          18     2.95  0.05
5  GO:0009579                               thylakoid         9           3     1.02  0.07
6  GO:0031984                organelle subcompartment         5           2     0.57  0.10
7  GO:0043227              membrane-bounded organelle       297          38    33.68  0.13
8  GO:0043231 intracellular membrane-bounded organelle       297          38    33.68  0.13
9  GO:0045298                         tubulin complex         7           2     0.79  0.18
10 GO:0044427                         chromosomal part        25          17     2.84  0.29
11 GO:0031981                            nuclear lumen        68           9     7.71  0.35
12 GO:0009505                     plant-type cell wall        11           2     1.25  0.36
13 GO:0043233                          organelle lumen        70           9     7.94  0.39
14 GO:0070013            intracellular organelle lumen        70           9     7.94  0.39
15 GO:0031974                  membrane-enclosed lumen        70           9     7.94  0.39
```

```
goResults[["BP"]]
```

```
        GO.ID                                        Term Annotated Significant Expected re
1  GO:0006334                           nucleosome assembly        23          16     2.74 3.
2  GO:0051322                                    anaphase        15          10     1.79 5.
3  GO:0051567                     histone H3-K9 methylation        16          10     1.91 1.
4  GO:0016572                     histone phosphorylation        11           8     1.31 3.
5  GO:0008283                          cell proliferation        29          15     3.45 6.
6  GO:0000911              cytokinesis by cell plate formation        26          12     3.10 7.
7  GO:0000079 regulation of cyclin-dependent protein k...         5           5     0.60 2.
8  GO:0051225                             spindle assembly         6           5     0.71 0.
9  GO:0007018                   microtubule-based movement        12           7     1.43 0.
10 GO:0010583                   response to cyclopentenone        10           6     1.19 0.
11 GO:0000087                   M phase of mitotic cell cycle        10           6     1.19 0.
12 GO:0010389 regulation of G2/M transition of mitotic...         5           4     0.60 0.
13 GO:0006275                 regulation of DNA replication         8           5     0.95 0.
14 GO:0000226       microtubule cytoskeleton organization        23          12     2.74 0.
15 GO:0006306                              DNA methylation        13           6     1.55 0.
```

3.8 Heatmap

The code to produce the heatmap (Supplementary Figure 9b), exactly as before.

```
pixelHeatmap <- function( m, zlim=range(m), col=colorRampPalette(c("blue","orange"))(100),
      pxWidth=1, pxHeight=1 ) {
   col <- col2rgb(col)/255
   mn <- ( m - zlim[1] ) / ( zlim[2] - zlim[1] )
   mn[ mn<0 ] <- 0
   mn[ mn>1 ] <- 1
   mn <- 1 + round( mn * (ncol(col)-1) )
   a <- array( NA_real_, c( nrow(m)*pxWidth, ncol(m)*pxHeight, 3 ) )
   for( i in 1:nrow(mn) )
      for( j in 1:ncol(mn) )
         a[ (((i-1)*pxWidth)+1) : (i*pxWidth), (((j-1)*pxHeight)+1) : (j*pxHeight),  ] <-
            rep( col[ , mn[ i, j ] ] , each = pxWidth*pxHeight )
   Image( a, colormode="color" )
}

relSig <- log2RelExprAt[ sig, ]
relSig[ relSig < -4 ] <- -4
ord <- hclust(dist(relSig))$order
hmSig <- pixelHeatmap( t( relSig[ord,] ), zlim=c( -3, 3 ), pxWidth=30,
   col=colorRampPalette(c("blue","gray","red"))(100) )
```

This time, we use these GO terms:

```
someGOTerms <- c( "GO:0009684", "GO:0003777", "GO:0016538", "GO:0051322",
   "GO:0051567", "GO:0000911", "GO:0003677", "GO:0000786" )

unname( t( sapply( someGOTerms, function(x) toTable(GOTERM[ x ])[1,2:3] ) ) )
```

```
     [,1]         [,2]
[1,] "GO:0009684" "indoleacetic acid biosynthetic process"
[2,] "GO:0003777" "microtubule motor activity"
[3,] "GO:0016538" "cyclin-dependent protein kinase regulator activity"
[4,] "GO:0051322" "anaphase"
[5,] "GO:0051567" "histone H3-K9 methylation"
[6,] "GO:0000911" "cytokinesis by cell plate formation"
[7,] "GO:0003677" "DNA binding"
[8,] "GO:0000786" "nucleosome"
```

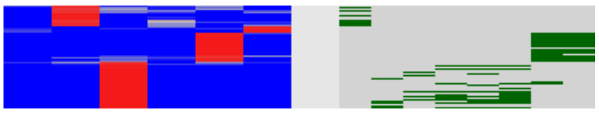
Make the seond heatmap, put it next to the first, and save the image.

```
sigInGO <- sapply( unname(someGOTerms), function(go)
   rownames(countsAt)[sig] %in% org.At.tairGO2TAIR[[ go ]] )

hmGO <- pixelHeatmap( t(sigInGO[ ord, ]), col=c( "lightgray", "darkgreen" ), pxWidth=20 )

spacer <- Image( array( .9, 30 * 3 * sum(sig), dim=c( 30, sum(sig), 3 ) ), colormode="color

writeImage(
   Image( abind::abind( hmSig, spacer, hmGO, along=1 ), colormode="color" ),
   files="heatmap_QC.png" )
```



Save the current state.

```
save.image( "QC_analysis_image.RData" )
```

4 Analysis of the mouse cells

4.1 Count table and normalization

The count table for the mouse data can be found in Supplementary Table 5:

```
dataMouse <- read.csv( "Supplementary_Table_5.csv", row.names=1 )
dataMouse[ 1:10, 1:5 ]
```

```
           length cell_01 cell_02 cell_03 cell_04
ERCC-00002   1061   13442   39379    6800    9697
ERCC-00003   1023     207       0      33     888
ERCC-00004    523    2762    6710    2526    2375
ERCC-00009    984       8    6125     217       5
ERCC-00012    994       0       0       0       0
ERCC-00013    808       0       0       0       0
ERCC-00014   1957       0       0       0       0
ERCC-00016    844       0       0       0       0
ERCC-00017   1136       0       0       0       0
ERCC-00019    644       0    1381       0       0
```

Again, we split the table into two sub-tables, one with the ERCC spikes, (countsERCC), one with the mouse genes (countsMmus). The first column, with the transcript length, is set aside.

```
geneTypes <- factor( c( Mm="Mmus", ER="ERCC" )[
   substr( rownames(dataMouse), 1, 2 ) ] )
countsMmus <- dataMouse[ which( geneTypes=="Mmus" ), -1 ]
countsERCC <- dataMouse[ which( geneTypes=="ERCC" ), -1 ]
lengthsMmus <- dataMouse[ which( geneTypes=="Mmus" ), 1 ]
lengthsERCC <- dataMouse[ which( geneTypes=="ERCC" ), 1 ]
```

Calculate size factors

```
sfMmus <- estimateSizeFactorsForMatrix( countsMmus )
sfERCC <- estimateSizeFactorsForMatrix( countsERCC )
rbind( sfMmus, sfERCC )
```

```
        cell_01   cell_02  cell_03   cell_04  cell_05  cell_08   cell_09  cell_10   cell_11
sfMmus 1.054806 0.8631698 1.314802 0.8392654 1.923025 2.038466 0.9484543 1.318534 0.9377162
sfERCC 1.341199 3.5780220 1.001075 1.0414057 1.258551 0.714875 0.5333393 1.178008 1.6108530
         cell_12    cell_13   cell_14  cell_15   cell_16   cell_17  cell_18   cell_19   cel
sfMmus 1.0650922 0.05676978 0.7071812 1.508444 1.5083955 1.6092835 1.348358 0.7929501 1.575
sfERCC 0.4885753 3.90732773 1.0575947 1.088776 0.3442378 0.9069057 1.019297 1.6010922 0.446
         cell_21   cell_22   cell_23   cell_24   cell_25    cell_26   cell_27   cell_28   c
sfMmus 1.1275932 0.7125047 0.2097335 1.2002740 1.2652195 0.09265965 1.6101646 1.5289089 0.6
sfERCC 0.9001471 1.9687150 1.6600806 0.5552053 0.7471373 3.73041540 0.9049005 0.9355409 2.7
         cell_30   cell_31   cell_32   cell_33   cell_34  cell_35   cell_36   cell_37   cel
sfMmus 0.8833294 1.6117294 0.9131230 0.6266095 0.4910416 1.231094 0.9456337 0.4741488 1.626
sfERCC 0.2198629 0.6530132 0.1959819 1.4602831 3.7636827 1.771898 0.1028514 2.7608761 0.694
        cell_39   cell_40   cell_41  cell_42   cell_43  cell_44   cell_45  cell_46   cell_4
sfMmus 2.067016 1.5703639 2.2319662 1.531569 0.4751253 1.433638 1.4033780 1.509945 1.385977
sfERCC 1.936887 0.3855543 0.7174266 1.316699 3.2065296 0.851406 0.3767175 1.511495 0.422995
         cell_48   cell_49  cell_50  cell_51   cell_52  cell_53  cell_54   cell_55  cell_56
sfMmus 0.9752096 1.7122014 1.395363 2.029381 0.4192704 1.733174 2.007804 1.6556897 1.551027
sfERCC 0.4137318 0.3755807 2.453767 2.088099 1.8768628 1.515678 1.754392 0.8157246 0.307815
         cell_57   cell_58   cell_59   cell_60  cell_61   cell_62   cell_63    cell_64  cel
sfMmus 1.8791183 1.8109205 2.0053160 0.5212242 1.286422 1.7020164 2.2123285  0.1373123 1.71
sfERCC 0.7091403 0.6844449 0.7593432 0.7342300 1.234711 0.4970688 0.9522804 13.6460462 1.62
        cell_66   cell_67  cell_68   cell_69   cell_70   cell_71   cell_72  cell_73   cell_
sfMmus 1.725958 1.4598322 1.674938 1.5899826 1.4466890 1.0997875 0.8398078 1.734892 1.23888
sfERCC 2.499659 0.7241718 1.702510 0.4682272 0.4076763 0.1952154 6.2377984 1.656883 0.85136
         cell_76   cell_77   cell_78   cell_79  cell_80   cell_81  cell_82  cell_83   cell_
sfMmus 0.9557100 1.2679037 0.7024893 1.6270177 1.509954 1.6773127 1.875683 0.633388 2.58677
sfERCC 0.6596151 0.5664658 1.1481714 0.8049232 6.085711 0.5256015 2.378635 3.469949 0.90372
         cell_85  cell_86   cell_87  cell_88  cell_89  cell_90   cell_91  cell_94 cell_95
sfMmus 0.3708505 1.964748 1.7201196 2.022113 1.324386 1.131953 0.9803028 1.369537 1.71133 1
sfERCC 3.1826913 1.139850 0.9348608 1.020459 1.204492 1.308119 0.9468956 2.970329 1.65396 0
```

Normalize by them:

```
nCountsERCC <- t( t(countsERCC) / sfERCC )
nCountsMmus <- t( t(countsMmus) / sfMmus )
```

We calculate the sample moments:

```
meansERCC <- rowMeans( nCountsERCC )
varsERCC <- rowVars( nCountsERCC )
cv2ERCC <- varsERCC / meansERCC^2

meansMmus <- rowMeans( nCountsMmus )
varsMmus <- rowVars( nCountsMmus )
cv2Mmus <- varsMmus / meansMmus^2
```

Normalize the mean counts by transcrip length (i.e., "per kilobase", "PK"), too:

```
meansERCCPK <- meansERCC / lengthsERCC * 1e3
meansMmusPK <- meansMmus / lengthsMmus * 1e3
```

4.2 Fit technical noise

We perform the fit as usual. However, as we have only rather few spikes, we have to be a bit more generous with the mean cut-off, now using the 80-percentile instead of the 95-percentile.

```
minMeanForFitA <- unname( quantile( meansERCC[ which( cv2ERCC > .3 ) ], .8 ) )
useForFitA <- meansERCC >= minMeanForFitA
minMeanForFitA
table( useForFitA )
```

```
[1] 81.63606
useForFitA
FALSE  TRUE
   71    21
```

Afterwards, we will compare with a fit using length-normalized counts. We prepare by finding the minimum for these, too:

```
minMeanForFitB <- unname( quantile( meansERCCPK[ which( cv2ERCC > .3 ) ], .8 ) )
useForFitB <- meansERCCPK >= minMeanForFitB
minMeanForFitB
table( A=useForFitA, B=useForFitB )
```

```
[1] 100.2707
       B
A       FALSE TRUE
  FALSE    70    1
  TRUE      1   20
```

Note that the two lists overlap well.

We perform both fits.

```
fitA <- glmgam.fit( cbind( a0 = 1, a1tilde = 1/meansERCC[useForFitA] ),
   cv2ERCC[useForFitA] )

fitB <- glmgam.fit( cbind( a0 = 1, a1tilde = 1/meansERCCPK[useForFitB] ),
   cv2ERCC[useForFitB] )
```

How much variance do the two fits explain?

```
residualA <- var( log( fitted.values(fitA) ) - log( cv2ERCC[useForFitA] ) )
```

```
totalA <- var( log( cv2ERCC[useForFitA] ) )

residualB <- var( log( fitted.values(fitB) ) - log( cv2ERCC[useForFitB] ) )
totalB <- var( log( cv2ERCC[useForFitB] ) )

# explained variances of log CV^2 values
c( A = 1 - residualA / totalA,
   B = 1 - residualB / totalB )
```

```
        A         B 
0.7945620 0.8873808 
```

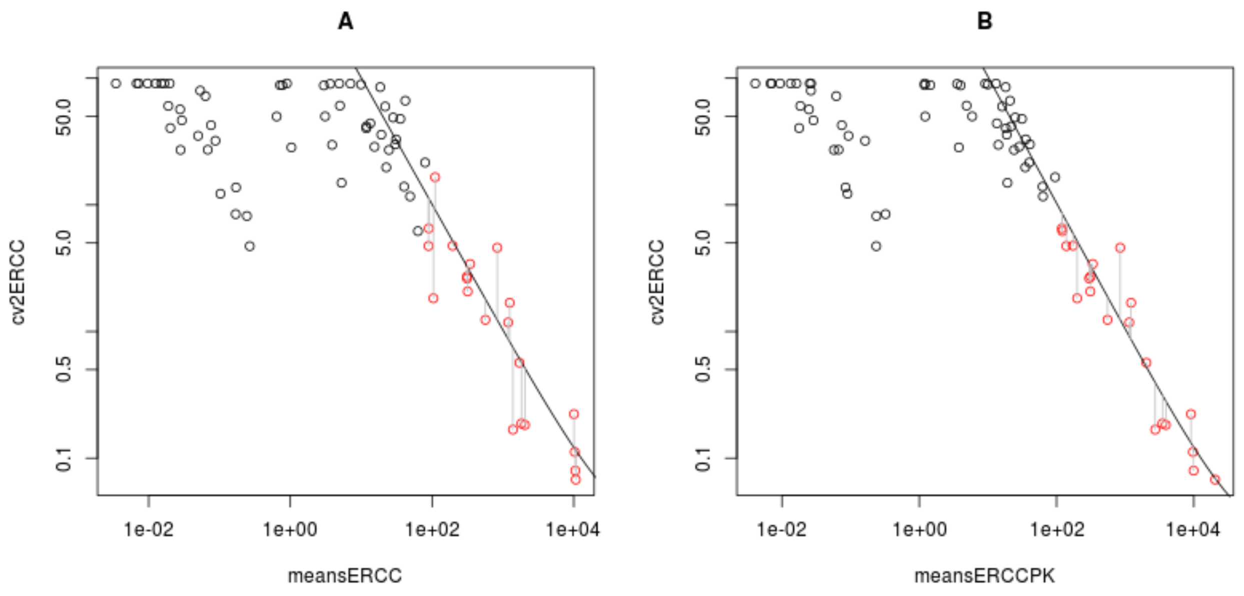
Fit B, which used the length-normalized counts, performed better.

As a second check, we plot both fits.

```
par( mfrow=c(1,2) )

plot( meansERCC, cv2ERCC, log="xy", col=1+useForFitA, main="A" )
xg <- 10^seq( -3, 5, length.out=100 )
lines( xg, coefficients(fitA)["a0"] + coefficients(fitA)["a1tilde"]/xg )
segments( meansERCC[useForFitA], cv2ERCC[useForFitA],
   meansERCC[useForFitA], fitA$fitted.values, col="gray" )

plot( meansERCCPK, cv2ERCC, log="xy", col=1+useForFitB, main="B" )
lines( xg, coefficients(fitB)["a0"] + coefficients(fitB)["a1tilde"]/xg )
segments( meansERCCPK[useForFitB], cv2ERCC[useForFitB],
   meansERCCPK[useForFitB], fitB$fitted.values, col="gray" )
```



4.3 Test for high variance

We start with the test using fit A:

```
minBiolDisp <- .5^2

xi <- mean( 1 / sfERCC )
m <- ncol(countsMmus)
psia1thetaA <- mean( 1 / sfERCC ) +
   ( coefficients(fitA)["a1tilde"] - xi ) * mean( sfERCC / sfMmus )
cv2thA <- coefficients(fitA)["a0"] + minBiolDisp + coefficients(fitA)["a0"] * minBiolDisp
```

```
testDenomA <- ( meansMmus * psia1thetaA + meansMmus^2 * cv2thA ) / ( 1 + cv2thA/m )

pA <- 1 - pchisq( varsMmus * (m-1) / testDenomA, m-1 )
padjA <- p.adjust( pA, "BH" )

table( padjA < .1 )
```

```
FALSE  TRUE
29489  1198
```

Using fit B and the length-normalized counts, we get

```
varsMmusPK <- rowVars( nCountsMmus / lengthsMmus * 1e3 )

psia1thetaB <- mean( 1 / sfERCC ) +
   ( coefficients(fitB)["a1tilde"] - xi ) * mean( sfERCC / sfMmus )
cv2thB <- coefficients(fitB)["a0"] + minBiolDisp + coefficients(fitB)["a0"] * minBiolDisp
testDenomB <- ( meansMmusPK * psia1thetaB + meansMmusPK^2 * cv2thB ) / ( 1 + cv2thB/m )

pB <- 1 - pchisq( varsMmusPK * (m-1) / testDenomB, m-1 )
padjB <- p.adjust( pB, "BH" )

table( B = padjB < .1 )
```

```
B
FALSE  TRUE
30137   523
```

Despite the better technical noise fit in case B (length adjusted), we get more results in case A (not length adjusted). The next section explores this.

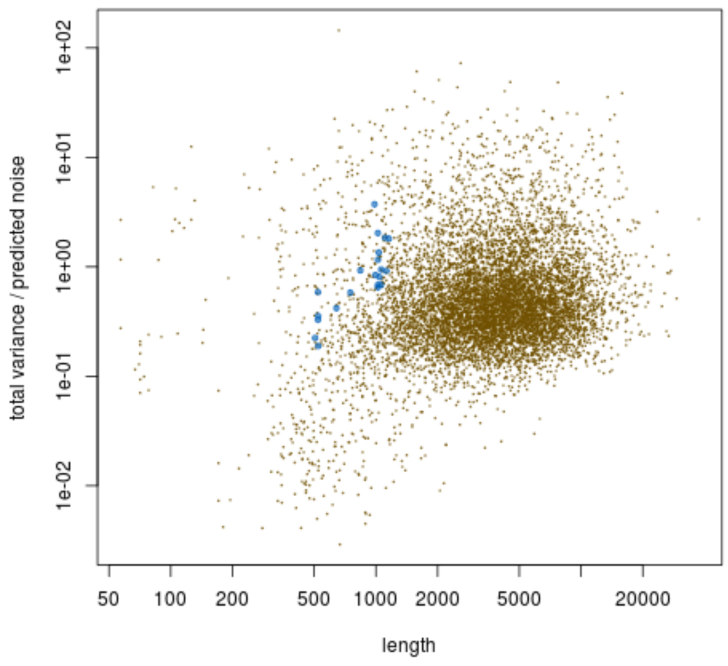
4.4 A diagnostic plot

We have a closer look at the reliability of the technical noise predictions from fit A. The variance from technical noise, predicted for a biological gene is given by Omega(0, mu), where mu is the normalized mean count for the gene, and Omega is the function defined in the Online Methods. Dividing by mu^2 to get CV^2 values (and ignoring the negligible term a0/m), we get

```
predictedNoiseCV2 <- psia1thetaA / meansMmus + coefficients(fitA)["a0"]
```

We plot the ratio of observed total CV^2 to predicted technical CV^{2}against transcript length, using only genes with a mean count above the cut-off also used for the fit. This is Supplementary Figure 7.

```
useInPlot <- meansMmus>minMeanForFitA
plot( lengthsMmus[useInPlot], ( cv2Mmus / predictedNoiseCV2 )[useInPlot], log="xy",
   pch=20, cex=.2, col = "#705000A0", xlab = "length", ylab="total variance / predicted noi
points( lengthsERCC[useForFitA], cv2ERCC[useForFitA] / fitted.values(fitA), pch=20, cex=1, 
```



See Supplementary Note 5 for a discussion.

4.5 Plot of results

To produce Figure 3, which depicts the results of fit and test A, the following code was used

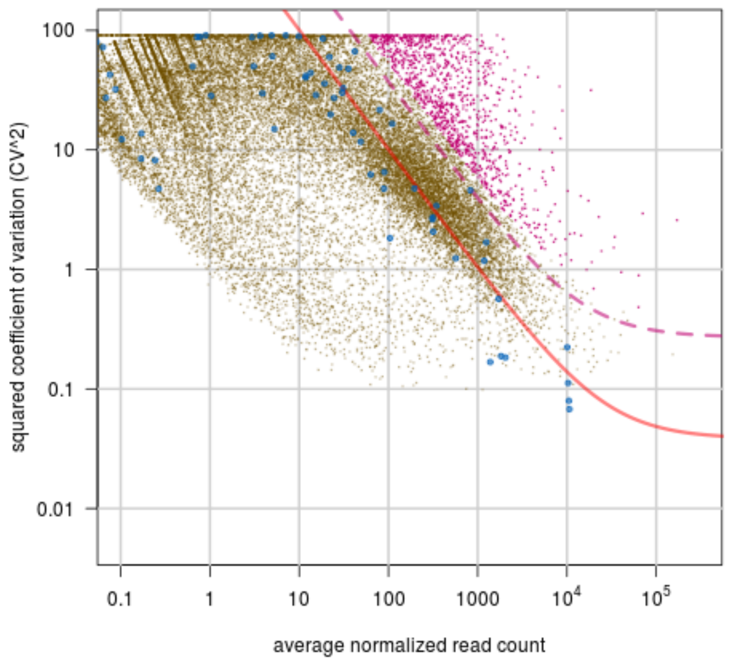
```
plot( NULL, xaxt="n", yaxt="n",
     log="xy", xlim = c( 1e-1, 3e5 ), ylim = c( .005, 100 ),
    xlab = "average normalized read count", ylab = "squared coefficient of variation (CV^2)
axis( 1, 10^(-1:5), c( "0.1", "1", "10", "100", "1000",
                   expression(10^4), expression(10^5) ) )
axis( 2, 10^(-2:2), c( "0.01", "0.1", "1", "10" ,"100"), las=2 )
abline( h=10^(-2:1), v=10^(-1:5), col="#D0D0D0", lwd=2 )

# Plot the plant genes, use a different color if they are highly variable
points( meansMmus, cv2Mmus, pch=20, cex=.2,
      col = ifelse( padjA < .1, "#C0007090", "#70500040" ) )

# Add the technical noise fit, as before
xg <- 10^seq( -2, 6, length.out=1000 )
lines( xg, coefficients(fitA)["a1tilde"] / xg + a0, col="#FF000080", lwd=3 )

# Add a curve showing the expectation for the chosen biological CV^2 thershold
lines( xg, psia1thetaA/xg + coefficients(fitA)["a0"] + minBiolDisp,
   lty="dashed", col="#C0007090", lwd=3 )

# Add the normalised ERCC points
points( meansERCC, cv2ERCC, pch=20, cex=1, col="#0060B8A0" )
```



Save image:

```
save.image( "mouse_analysis_image.RData" )
```

Date: 2013-08-08 11:03:10 CEST

Author: Simon Anders

Org version 7.8.02 with Emacs version 23

Validate XHTML 1.0