HTSFilter: Independent data-based filtering for replicated transcriptome sequencing experiments

Andrea Rau, Mélina Gallopin, Gilles Celeux, Florence Jaffrézic

Modified: March 27, 2013. Compiled: March 27, 2013

Abstract

This vignette illustrates the use of the *HTSFilter* package to filter replicated data from transcriptome sequencing experiments (e.g., RNA sequencing data) for a variety of different data classes: *matrix*, *data.frame*, *CountDataSet* (the S4 class associated with the *DESeq* package), and the S3 classes associated with the *edgeR* package (*DGEList*, *DGEExact*, *DGEGLM*, and *DGELRT*).

Contents

1 Introduction

High-throughput sequencing (HTS) data, such as RNA-sequencing (RNA-seq) data, are increasingly used to conduct differential analyses, in which statistical tests are performed for each biological feature (e.g., a gene, transcript, exon) in order to identify those

whose expression levels show systematic covariation with a particular condition, such as a treatment or phenotype of interest. For the remainder of this vignette, we will focus on gene-level differential analyses, although these methods may also be applied to differential analyses of (count-based measures of) transcript- or exon-level expression.

Because hypothesis tests are performed for gene-by-gene differential analyses, the obtained p-values must be adjusted to correct for multiple testing. However, procedures to adjust p-values to control the number of detected false positives often lead to a loss of power to detect truly differentially expressed (DE) genes due to the large number of hypothesis tests performed. To reduce the impact of such procedures, independent data filters are often used to identify and remove genes that appear to generate an uninformative signal [**?**]; this in turn moderates the correction needed to adjust for multiple testing. For independent filtering methods for microarray data, see for example the *genefilter* Bioconductor package [**?**].

The *HTSFilter* package implements a novel independent filtering procedure based on the calculation of a similarity index among biological replicates for read counts arising from replicated transcriptome sequencing (RNA-seq) data. This technique provides an intuitive data-driven way to filter high-throughput transcriptome sequencing data and to effectively remove genes with low, constant expression levels without incorrectly removing those that would otherwise have been identified as DE. The two fundamental assumptions of the filter implemented in the *HTSFilter* package are as follows:

1. Biological replicates are present for each experimental condition,

2. Data can be appropriately normalized (scaled) to correct for systematic inter-sample biases, and

Assuming these conditions hold, *HTSFilter* implements a method to identify a filtering threshold that maximizes the *filtering similarity* among replicates, that is, one where most genes tend to either have normalized counts less than or equal to the cutoff in all samples (i.e., filtered genes) or greater than the cutoff in all samples (i.e., non-filtered genes). This filtering similarity is defined using the global Jaccard index, that is, the average Jaccard index calculated between pairs of replicates within each experimental condition; see **?**] for more details.

For more information about between-sample normalization strategies, see [**?**]; in particular, strategies for normalizing data with differences in library size and composition may be found in [**?**] and [**?**], and strategies for normalizing data exhibiting sample-specific biases due to GC content may be found in [**?**] and [**?**]. Within the *HTSFilter* package, the Trimmed Means of M-values (TMM) [**?**] and DESeq [**?**] normalization strategies may be used prior to calculating an appropriate data-based filter. If an alternative normalization strategy is needed or desired, the normalization may be applied prior to filtering the data with `normalization="none"` in the `HTSFilter` function; see Section 6 for an example.

The *HTSFilter* package is able to accommodate unnormalized or normalized replicated count data in the form of a *matrix* or *data.frame* (in which each row corresponds

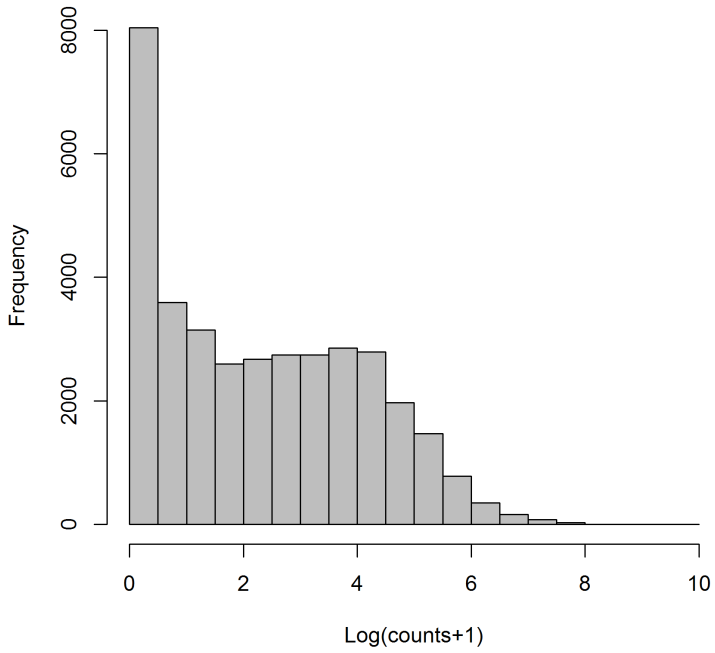

Figure 1: Histogram of log transformed counts from the Sultan et al. data [**?**], illustrating the large number of genes with very small counts as well as the large heterogeneity in counts observed.

to a biological feature and each column to a biological sample), a *CountDataSet* (the S4 class associated with the *DESeq* package), or one of the S3 classes associated with the *edgeR* package (*DGEList*, *DGEExact*, *DGEGLM*, and *DGELRT*), as illustrated in the following sections.

Finally, we note that the filtering method implemented in the *HTSFilter* package is designed to filter transcriptome sequencing, and not microarray, data; in particular, the proposed filter is effective for data with features that take on values over a large order of magnitude and with a subset of features exhibiting small levels of expression across samples (see, for example, Figure 1). In this vignette, we illustrate its use on count-based measures of gene expression, although its use is not strictly limited to discrete data.

2 Input data

For the purposes of this vignette, we make use of data from a study of sex-specific expression of liver cells in human. Sultan et al. [**?**] obtained a high-throughput sequencing data (using a 1G Illumina Genome Analyzer sequencing machine) from a

human embryonic kidney and a B cell line, with two biological replicates each. The raw read counts and phenotype tables were obtained from the ReCount online resource [**?**].

To begin, we load the *HTSFilter* package, and attach the gene-level count data contained in `sultan`:

```
> library(HTSFilter)
> data("sultan")
> hist(log(exprs(sultan)+1), col="grey", breaks=25, main="",
+   xlab="Log(counts+1)")
> pData(sultan)
```

```
          sample.id num.tech.reps    cell.line
SRX008333 SRX008333             1 Ramos B cell
SRX008334 SRX008334             1 Ramos B cell
SRX008331 SRX008331             1      HEK293T
SRX008332 SRX008332             1      HEK293T
```

```
> dim(sultan)
```

```
Features  Samples
    9010        4
```

The unfiltered data contain 9010 genes in four samples (two replicates per condition).

3 *matrix* and *data.frame* classes

To filter high-throughput sequencing data in the form of a *matrix* or *data.frame*, we first access the expression data, contained in `exprs(sultan)`, and create a vector identifying the condition labels for each of the samples via the `pData` Biobase function. We then filter the data using the `HTSFilter` function, specifying that the number of tested thresholds be only 25 (`s.len=25`) rather than the default value of 100 to reduce computation time for this example. Note that as it is unspecified, the default normalization method is used for filtering the data, namely the Trimmed Mean of M-values (TMM) method of Robinson and Oshlack [**?**]. To use the DESeq normalization method [**?**], `normalization="DESeq"` may be specified.

```
> mat <- exprs(sultan)
> conds <- pData(sultan)$cell.line
> ## Only 25 tested thresholds to reduce computation time
> filter <- HTSFilter(mat, conds, s.len=25)
> mat <- filter$filteredData
> dim(mat)
```

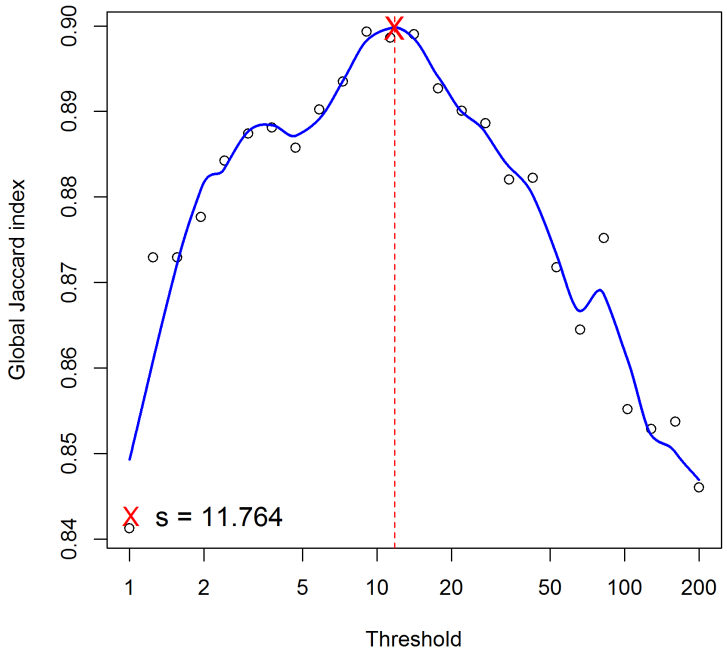

Figure 2: Global Jaccard index for the `sultan` data calculated for a variety of threshold values after TMM normalization [**?**], with a loess curve (blue line) superposed and data-based threshold values (red cross and red dotted line) equal to 11.764.

```
[1] 4995    4
```

```
> dim(filter$removedData)
```

```
[1] 4015    4
```

For this example, we find a data-based threshold equal to 11.764; genes with normalized values less than this threshold in all samples are filtered from subsequent analyses. The proposed filter thus removes 4015 genes from further analyses, leaving 4995 genes.

We note that an important part of the filter proposed in the *HTSFilter* package is a check of the behavior of the global similarity index calculated over a range of threshold values, and in particular, to verify that a reasonable maximum value is reached for the global similarity index over the range of tested threshold values (see Figure 2); the maximum possible value for the global Jaccard index is 1. To illustrate the importance of this check, we attempt to re-apply the proposed filter to the previously filtered data (in practice, of course, this would be nonsensical):

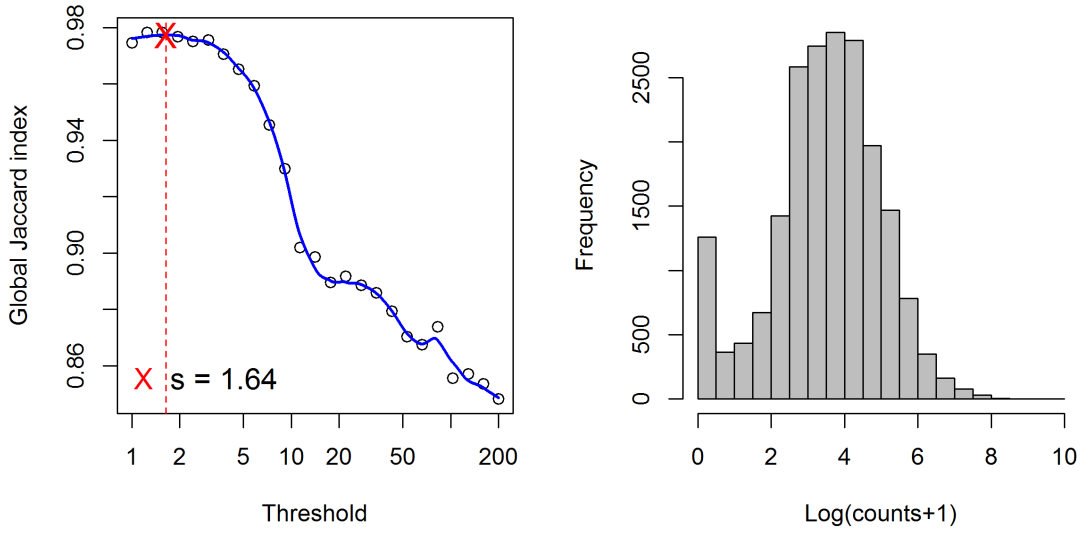

Figure 3: (left) Global Jaccard index for the `sultan` data calculated for a variety of threshold values after TMM normalization [**?**], with a loess curve (blue line) superposed and data-based threshold values (red cross and red dotted line) equal to 11.764. (right) Global Jaccard index for the previously filtered `sultan` data, with loess curve (blue line) superposed as before.

```
> par(mfrow = c(1,2), mar = c(4,4,2,2))
> filter.2 <- HTSFilter(mat, conds, s.len=25)
> dim(filter.2$removedData)
```

```
[1] 0 4
```

```
> hist(log(filter.2$filteredData+1), col="grey", breaks=25, main="",
+   xlab="Log(counts+1)")
```

In the lefthand panel of Figure 3, we note a plateau of large global Jaccard index values for thresholds less than 2, with a decrease thereafter; this corresponds to filtering no genes, unsurprising given that genes with low, constant levels of expression have already been filtered from the analysis (see the righthand panel of Figure 3).

4 *DESeq* package pipeline: S4 class *CountDataSet*

The *HTSFilter* package allows for three potential applications of the proposed filter within the *DESeq* analysis pipeline:

1. Estimation of library sizes and dispersion parameters (the `estimateSizeEffects` and `estimateDispersions` functions in *DESeq*), followed by data filtering on normalized data (recommended);

2. Estimation of library sizes (`estimateSizeEffects`), data filtering on normalized data, and estimation of dispersion parameters (`estimateDispersions`);

3. Data filtering on normalized data, followed by re-estimation of library sizes and estimation of dispersion parameters (`estimateSizeEffects` and `estimateDispersions`).

We note that the primary difference among the three strategies would be seen in the dispersion parameters estimates for genes with low levels of expression; because fitted dispersion values are estimated based on the mean-dispersion relationship observed across the full data in the *DESeq* package, estimates obtained on filtered data will necessarily be slightly different from those obtained on unfiltered data, as genes with low levels of expression would have been removed from the former. On the other hand, we note that the filtering thresholds (and as such, the genes filtered from the analysis) are identical for the three strategies listed above. The estimated library sizes may differ slightly in the third strategy as compared to the first and second strategies; however, this difference will be minimal as only genes with weak, constant levels of expression are filtered from the analysis. A full discussion of these three strategies is beyond the scope of this vignette; however, in practice we recommend the use of the first strategy: estimation of library sizes and dispersion parameters prior to data filtering.

To filter high-throughput sequencing data in the form of a *CountDataSet* (the class used within the *DESeq* pipeline for differential analysis), we coerce `sultan` into an object of the class *CountDataSet*. Once again, we specify that the number of tested thresholds be only 25 (`s.len=25`) rather than the default value of 100 to reduce computation time. For objects in the form of a *CountDataSet*, the default normalization strategy is `"DESeq"`, although alternative normalization strategies may also be used.

```
> cds <- newCountDataSet(exprs(sultan), conds)
> cds <- estimateSizeFactors(cds)
> cds <- estimateDispersions(cds)
> ## Only 25 tested thresholds to reduce computation time
> cds <- HTSFilter(cds, s.len=25)$filteredData
> res <- nbinomTest(cds, levels(conds)[1], levels(conds)[2])
> class(cds)

[1] "CountDataSet"
attr(,"package")
[1] "DESeq"

> dim(cds)

Features  Samples
    5143        4
```

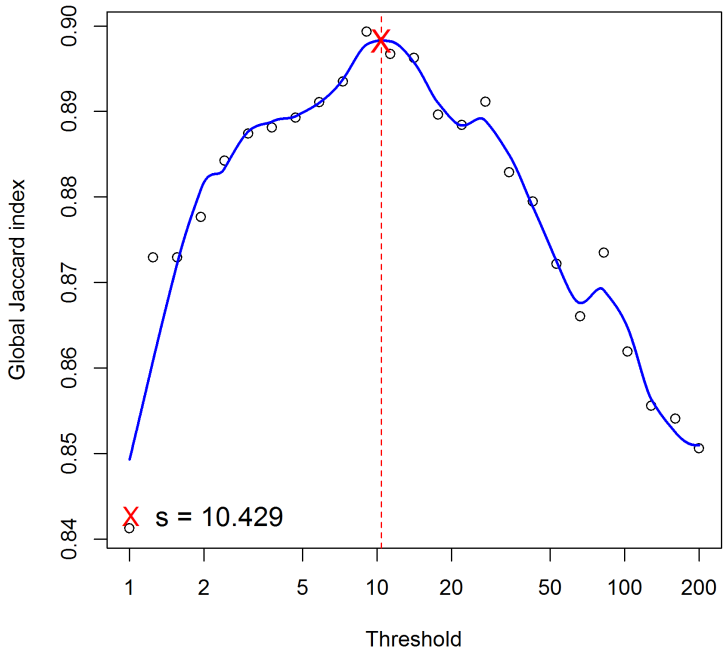

Figure 4: Global Jaccard index for the `sultan` data calculated for a variety of threshold values after DESeq normalization [**?**], with a loess curve (blue line) superposed and data-based threshold values (red cross and red dotted line) equal to 10.429.

As the normalization strategy used here was slightly different, the proposed filter now removes 3867 genes from further analyses, leaving 5143 genes. Again we verify the behavior of the global similarity index calculated over a range of threshold values (see Figure 4). For this example, we find a data-based threshold equal to 10.429; genes with normalized values less than this threshold in all samples are filtered from subsequent analyses.

5 *edgeR* package pipeline

We next illustrate the use of *HTSFilter* within the *edgeR* pipeline for differential analysis (S3 classes *DGEList*, *DGEExact*, *DGEGLM*, or *DGELRT*). For the purposes of this vignette, we will consider the S3 classes *DGEExact* and *DGELRT*. The former is the class containing the results of the differential expression analysis between two groups of count libraries (resulting from a call to the function `exactTest` in *edgeR*); the latter is the class containing the results of a generalized linear model (GLM)-based differential analysis (resulting from a call to the function `glmLRT` in *edgeR*). Although the filter may

be applied earlier in the *edgeR* pipeline (i.e., to objects of class *DGEList* or *DGEGLM*), we do not recommend doing so, as parameter estimation makes use of counts adjusted using a a quantile-to-quantile method (pseudo-counts).

5.1 S3 class *DGEExact*

We first coerce the data into the appropriate class with the function `DGEList`, where the `group` variable is set to contain a vector of condition labels for each of the samples. Next, after calculating normalizing factors to scale library sizes (`calcNormFactors`), we estimate common and tagwise dispersion parameters using `estimateCommonDisp` and `estimateTagwiseDisp` and obtain differential analysis results using `exactTest`. Finally, we apply the filter using the `HTSFilter` function, again specifying that the number of tested thresholds be only 25 (`s.len=25`) rather than the default value of 100. Note that as it is unspecified, the default normalization method is used for filtering the data, namely the Trimmed Mean of M-values (TMM) method [**?**]; alternative normalization, including `"pseudo.counts"` for the quantile-to-quantile adjusted counts used for parameter estimation, may also be specified. We suppress the plot of the global Jaccard index using `plot = FALSE`, as it is identical to that shown in Figure 2.

```
> dge <- DGEList(counts=exprs(sultan), group=conds)
> dge <- calcNormFactors(dge)
> dge <- estimateCommonDisp(dge)
> dge <- estimateTagwiseDisp(dge)
> et <- exactTest(dge)
> et <- HTSFilter(et, DGEList=dge, s.len=25, plot=FALSE)$filteredData
> dim(et)

[1] 4995    3

> class(et)

[1] "DGEExact"
attr(,"package")
[1] "edgeR"

> topTags(et)

Comparison of groups:  Ramos B cell-HEK293T
                     logFC   logCPM        PValue
ENSG00000133124 -14.395299 11.56904  0.000000e+00
ENSG00000105369  11.925763 11.22543  0.000000e+00
ENSG00000135144  10.923026 11.05287 5.303498e-319
ENSG00000111348  13.798322 10.91659 5.749762e-306
```

```
ENSG00000177606  -9.732512 10.81547 5.364706e-280
ENSG00000118308  13.492720 10.60871 1.204688e-270
ENSG00000100721  14.402790 11.53678 8.233173e-264
ENSG00000012124  10.171960 10.28356 8.667924e-264
ENSG00000046604 -12.906072 10.11017 4.786346e-226
ENSG00000110777  13.854362 10.98334 2.190549e-220
                         FDR
ENSG00000133124  0.000000e+00
ENSG00000105369  0.000000e+00
ENSG00000135144 8.830325e-316
ENSG00000111348 7.180016e-303
ENSG00000177606 5.359342e-277
ENSG00000118308 1.002903e-267
ENSG00000100721 5.412035e-261
ENSG00000012124 5.412035e-261
ENSG00000046604 2.656422e-223
ENSG00000110777 1.094179e-217
```

Note that the filtered data are of the class *DGEExact*, allowing for a call to the `topTags` function.

```
> topTags(et)

Comparison of groups:  Ramos B cell-HEK293T
                     logFC   logCPM        PValue
ENSG00000133124 -14.395299 11.56904  0.000000e+00
ENSG00000105369  11.925763 11.22543  0.000000e+00
ENSG00000135144  10.923026 11.05287 5.303498e-319
ENSG00000111348  13.798322 10.91659 5.749762e-306
ENSG00000177606  -9.732512 10.81547 5.364706e-280
ENSG00000118308  13.492720 10.60871 1.204688e-270
ENSG00000100721  14.402790 11.53678 8.233173e-264
ENSG00000012124  10.171960 10.28356 8.667924e-264
ENSG00000046604 -12.906072 10.11017 4.786346e-226
ENSG00000110777  13.854362 10.98334 2.190549e-220
                         FDR
ENSG00000133124  0.000000e+00
ENSG00000105369  0.000000e+00
ENSG00000135144 8.830325e-316
ENSG00000111348 7.180016e-303
ENSG00000177606 5.359342e-277
ENSG00000118308 1.002903e-267
ENSG00000100721 5.412035e-261
```

```
ENSG00000012124 5.412035e-261
ENSG00000046604 2.656422e-223
ENSG00000110777 1.094179e-217
```

5.2 S3 class *DGELRT*

We follow the same steps as the previous example, where the `estimateGLMCommonDisp`, `estimateGLMTrendedDisp`, and `estimateGLMTagwiseDisp` functions are now used to obtain per-gene dispersion parameter estimates, the `glmFit` function is used to fit a negative binomial generalized log-linear model to the read counts for each gene, and the `glmLRT` function is used to conduct likelihood ratio tests for one or more coefficients in the GLM. The output of `glmLRT` is an S3 object of class *DGELRT* and contains the GLM differential analysis results. As before, we apply the filter using the `HTSFilter` function, again suppressing the plot of the global Jaccard index using `plot = FALSE`, as it is identical to that shown in Figure 2.

```
> design <- model.matrix(~conds)
> dge <- DGEList(counts=exprs(sultan), group=conds)
> dge <- calcNormFactors(dge)
> dge <- estimateGLMCommonDisp(dge,design)
> dge <- estimateGLMTrendedDisp(dge,design)
> dge <- estimateGLMTagwiseDisp(dge,design)
> fit <- glmFit(dge,design)
> lrt <- glmLRT(fit,coef=2)
> lrt <- HTSFilter(lrt, DGEGLM=fit, s.len=25, plot=FALSE)$filteredData
> dim(lrt)

[1] 4995    4

> class(lrt)

[1] "DGELRT"
attr(,"package")
[1] "edgeR"
```

Note that the filtered data are of the class *DGEList*, allowing for a call to the `topTags` function.

```
> topTags(lrt)

Coefficient:  condsRamos B cell
                     logFC   logCPM       LR        PValue
ENSG00000133124 -14.394460 11.56767 1569.700  0.000000e+00
ENSG00000105369  11.925772 11.22727 1645.374  0.000000e+00
```

```
ENSG00000135144  10.921894 11.05362 1496.354  0.000000e+00
ENSG00000111348  13.797195 10.91711 1469.436 1.714408e-321
ENSG00000118308  13.494132 10.60839 1324.932 4.316647e-290
ENSG00000100721  14.399210 11.53641 1290.142 1.568521e-282
ENSG00000012124  10.171671 10.28368 1265.222 4.083826e-277
ENSG00000177606  -9.731883 10.81474 1265.068 4.410785e-277
ENSG00000110777  13.850422 10.98235 1100.755 2.263157e-241
ENSG00000149418  10.899579 10.18705 1071.703 4.667481e-235
                          FDR
ENSG00000133124  0.000000e+00
ENSG00000105369  0.000000e+00
ENSG00000135144  0.000000e+00
ENSG00000111348 2.140865e-318
ENSG00000118308 4.312331e-287
ENSG00000100721 1.305794e-279
ENSG00000012124 2.753984e-274
ENSG00000177606 2.753984e-274
ENSG00000110777 1.256052e-238
ENSG00000149418 2.331407e-232
```

6 Alternative normalization using *EDAseq*

As a final example, we illustrate the use of the *HTSFilter* package with an alternative normalization strategy, namely the full quantile normalization method in the *EDASeq* package; such a step may be useful when the TMM or DESeq normalization methods are not appropriate for a given dataset. Once again, we create a new object of the appropriate class with the function `newSeqExpressionSet` and normalize data using the `betweenLaneNormalization` function (with `which="full"`) in *EDASeq*.

```
> library(EDASeq)
> ses <- newSeqExpressionSet(exprs(sultan),
+         phenoData=pData(sultan))
> ses.norm <- betweenLaneNormalization(ses, which="full")
```

Subsequently, `HTSFilter` is applied to the normalized data (again using `s.len=25`), and the normalization method is set to `norm="none"`. We may then make use of the `on` vector in the results, which identifies filtered and unfiltered genes (respectively) with 0 and 1, to identify rows in the original data matrix to be retained.

```
> filter <- HTSFilter(exprs(ses.norm), conds, s.len=25, norm="none",
+           plot=FALSE)
> head(filter$on)
> table(filter$on)
```

7 Session Info

```
> sessionInfo()

R Under development (unstable) (2012-12-15 r61341)
Platform: x86_64-w64-mingw32/x64 (64-bit)

locale:
[1] LC_COLLATE=English_United States.1252
[2] LC_CTYPE=English_United States.1252
[3] LC_MONETARY=English_United States.1252
[4] LC_NUMERIC=C
[5] LC_TIME=English_United States.1252

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

other attached packages:
 [1] HTSFilter_0.99.6         DESeq_1.11.3
 [3] lattice_0.20-10          locfit_1.5-8
 [5] Biobase_2.19.1           BiocGenerics_0.5.6
 [7] edgeR_3.1.3              limma_3.15.5
 [9] SweaveListingUtils_0.5.5 startupmsg_0.7.2

loaded via a namespace (and not attached):
 [1] annotate_1.37.3      AnnotationDbi_1.21.9
 [3] DBI_0.2-5            genefilter_1.41.1
 [5] geneplotter_1.37.0   grid_3.0.0
 [7] IRanges_1.17.24      RColorBrewer_1.0-5
 [9] RSQLite_0.11.2       stats4_3.0.0
[11] survival_2.37-2      tools_3.0.0
[13] XML_3.95-0.1         xtable_1.7-0
```