Analyzing RNA-seq data for differential exon usage with the DEXSeq package

Alejandro Reyes, Simon Anders, Wolfgang Huber

2011-10-26

Contents

Abstract

RNA-seq is a powerful tool for transcriptome analysis. It enables the discovery of novel transcript splice sites and isoforms, and there is interest in the quantitative comparison of exon usage between different conditions. For the analysis of differential expression between conditions, appropriate modeling of the experimental and biological variability is important to have control over type I error, and such capabilities are offered, for instance, by the packages *edgeR* [3] and *DESeq* [1]. In this package, we provide a method to systematically detect differential exon usage using RNA-seq that takes into account this variability. We use as input the number of reads mapping to each of the exons of a genome. The method is demonstrated on the data from the package *pasilla*.

1 The Pasilla dataset

We will use the `pasillaExons` dataset from the *pasilla* package. `pasillaExons` is an object of class *ExonCountSet*. Brooks et al. [2] investigated the effect of siRNA knock-down of Pasilla, whose protein is known to bind to mRNA in the spliceosome, and which is thought to be involved in the regulation of splicing, on the transcriptome of fly S2-DRSC cells. Pasilla is the Drosophila melanogaster ortholog of mammalian NOVA1 and NOVA2. The dataset, which is provided by NCBI Gene Expression Omnibus (GEO) under the accession number GSE18508[1], contains 3 biological replicates of the knockdown as well as 4 biological replicates for the untreated control.

In the *pasilla* package, we chose a subset of genes in order to speed up the computations shown in this vignette. We start by loading the *DEXSeq* package and the example data.

```
> library("DEXSeq")
```

```
> data("pasillaExons", package = "pasilla")
```

If you have not yet installed the *pasilla* package, you can do so with

```
> source("http://www.bioconductor.org/biocLite.R")
> biocLite("pasilla")
```

The `design` accessor function shows the available sample annotations.

```
> design(pasillaExons)
```

```
             condition replicate        type
treated1fb     treated         1 single-read
treated2fb     treated         2  paired-end
treated3fb     treated         3  paired-end
untreated1fb untreated         1 single-read
untreated2fb untreated         2 single-read
untreated3fb untreated         3  paired-end
untreated4fb untreated         4  paired-end
```

We also print the first 6 lines of selected columns of the feature data annotation:

```
> head(fData(pasillaExons)[, c(1, 2, 9:12)])
```

```
                     geneID exonID   chr   start     end strand
FBgn0000256:001 FBgn0000256   E001 chr2L 3872658 3872947      -
FBgn0000256:002 FBgn0000256   E002 chr2L 3873019 3873322      -
FBgn0000256:003 FBgn0000256   E003 chr2L 3873385 3874395      -
FBgn0000256:004 FBgn0000256   E004 chr2L 3874450 3875302      -
FBgn0000256:005 FBgn0000256   E005 chr2L 3878895 3879067      -
FBgn0000256:006 FBgn0000256   E006 chr2L 3879652 3880038      -
```

There are 46 genes in the dataset, of these, there is one with 36 exons, and for instance, three with 16 exons:

```
> table(table(geneIDs(pasillaExons)))
```

[1] http://www.ncbi.nlm.nih.gov/projects/geo/query/acc.cgi?acc=GSE18508

```
    0     1     2     3     4     5     6     7     8     9    10    11    12
14424     2     3     3     2     4     2     3     3     3     2     3     2
   15    16    17    19    22    23    24    25    36
    1     3     2     1     1     3     1     1     1
```

In Section 8, we explain how you can create analogous data objects from your own data.

2 Normalisation and dispersion estimation

Different samples might be sequenced with different depths. In order to adjust for such coverage biases, we introduce size factor parameters. *DEXSeq* uses the same method as *DESeq*, which is provided in the function `estimateSizeFactors`.

```
> pasillaExons <- estimateSizeFactors(pasillaExons)
```

```
> sizeFactors(pasillaExons)

  treated1fb   treated2fb   treated3fb untreated1fb untreated2fb untreated3fb
       1.084        0.970        0.932        1.092        1.447        0.827
untreated4fb
       0.827
```

Then, to test for differential expression, we need to estimate the data's variance. This is needed in order to be able to distinguish between normal technical and biological variation (noise) and real effects on exon expression due to the different conditions. The information on the size of the noise is drawn from the biological replicates in the dataset. However, as typical for RNA-seq experiments, the number of replicates is too small to estimate variance or dispersion parameters individually exon by exon. Instead, variance information is shared across exons and genes, in an intensity dependent manner. Computationally, this is done through Cox-Reid likelihood estimation (our method follows that of the package *edgeR* [3]). These steps are implemented in the function `estimateExonDispersionsForModelFrame`. To create a data frame that encodes the model for a gene, with columns `sample`, `exon`, `condition`, `sizeFactors` and `count`, the function `modelFrameForGene` is used. The function `estimateExonDispersionsForModelFrame` inputs this data frame to make the CR dispersion estimates.

```
> head(modelFrameForGene(pasillaExons, "FBgn0010909"))

      sample exon sizeFactor condition replicate        type dispersion count
1 treated1fb E001       1.08   treated         1 single-read         NA  3423
2 treated1fb E002       1.08   treated         1 single-read         NA   443
3 treated1fb E003       1.08   treated         1 single-read         NA   589
4 treated1fb E004       1.08   treated         1 single-read         NA   671
5 treated1fb E005       1.08   treated         1 single-read         NA   637
6 treated1fb E006       1.08   treated         1 single-read         NA   816
```

```
> estimateExonDispersionsForModelFrame(modelFrameForGene(pasillaExons,
+     "FBgn0010909"))

   E001    E002    E003    E004    E005    E006    E007    E008    E009    E010
0.19954 0.05746 0.00727 0.02924 0.06475 0.02336 0.02607 0.01557 0.02675 0.02816
   E011    E012    E013    E014    E015    E016    E017    E018    E019    E020
```

```
0.04366 0.08122 0.02232 2.51611 0.09903 0.07736 0.05493 0.01601 0.22066 0.01108
   E021    E022    E023
1.43637 0.32581 0.04331
```

The function `estimateDispersions` provides an interface that makes a call to `estimateExonDispersionsForModelFrame` for each of the exons that will be tested. Before starting estimating the CR dispersion estimates, `estimateDispersions` will first define the ”testable” exons, meaning those exons that its total sum of counts over all the samples is higher than the parameter `minCount`, those genes that its number of exons is not higher than `maxExon` and those genes that have more than one ”testable” exon. The result from `estimateDispersions` is stored in the column `dispBeforeSharing` of the feature data.

```
> pasillaExons <- estimateDispersions(pasillaExons)
```

Then the function `fitDispersionFunction` its called, in which a dispersion-mean relation $\alpha(\mu) = \alpha_0 + \alpha_1/\mu$ is fitted to the individual CR dispersion values (dispersions before sharing), the coefficients are stored in the slot `dispFitCoefs` and finally, for each exon, the maximum betweem the dispersion before sharing and the fitted dispersion value is taken as the exon's final dispersion value and stored in the `dispersion` slot. Please take into consideration that this fit could be difficult to do in cases where the dispersion estimates are to big, this sometimes tends to break this function. In this special cases please do not hesitate to contact the developers.

```
> pasillaExons <- fitDispersionFunction(pasillaExons)
> head(fData(pasillaExons)$dispBeforeSharing)

[1] 0.0381 0.0130 0.0251 0.0453 0.0517 0.0846

> pasillaExons@dispFitCoefs

     (Intercept) I(1/means[good])
          0.0375           2.2664

> head(fData(pasillaExons)$dispFitted)

[1] 0.0547 0.0479 0.0410 0.0427 0.0559 1.0416

> head(fData(pasillaExons)$dispersion)

[1] 0.0547 0.0479 0.0410 0.0453 0.0559 1.0416
```

To make a fit diagnostic, each individual exon dispersion is plotted vs its mean expression value. The function we just fitted is plotted in the panel and passing through the data points. This gives a good diagnostic of the fit. Figure 1

```
> meanvalues <- rowMeans(counts(pasillaExons))
> plot(meanvalues, fData(pasillaExons)$dispBeforeSharing, log = "xy",
+     main = "mean vs CR dispersion")
> x <- 0.01:max(meanvalues)
> y <- pasillaExons@dispFitCoefs[1] + pasillaExons@dispFitCoefs[2]/x
> lines(x, y, col = "red")
```

In Section 4, we will see how to incorporate further experimental or technical variables into the dispersion estimation.

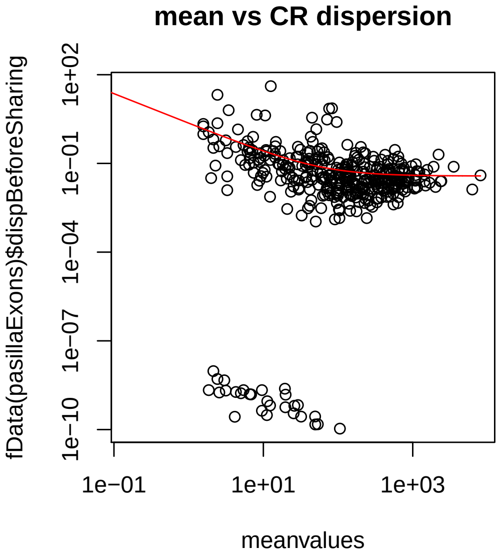

Figure 1: Fit diagnostics plot.

3 Testing for differential exon usage

Having the dispersion estimates and the size factors, we can now test for differential exon usage. For each gene, we fit a generalized linear model with the formula

```
sample + exon + condition * I(exon == exonID)
```

and compare it to the smaller model (the null model)

```
sample + exon + condition.
```

and the deviances of both fits are compared using a χ^2-distribution. The actual test (which already includes a call to `modelFrameForGene`) is performed by the function `testGeneForDEU`:

```
> testGeneForDEU(pasillaExons, "FBgn0010909")

     deviance df   pvalue
E001   0.0874  1 7.67e-01
E002   0.3779  1 5.39e-01
E003   0.0563  1 8.12e-01
E004   1.2049  1 2.72e-01
E005   2.3074  1 1.29e-01
E006   0.6921  1 4.05e-01
E007   0.2877  1 5.92e-01
E008   0.0324  1 8.57e-01
E009   0.0445  1 8.33e-01
E010  69.8529  1 1.11e-16
E011   1.5126  1 2.19e-01
```

```
E012   0.0438  1 8.34e-01
E013   1.4185  1 2.34e-01
E014   0.0812  1 7.76e-01
E015   0.0799  1 7.77e-01
E016   0.2061  1 6.50e-01
E017   0.3008  1 5.83e-01
E018   1.9700  1 1.60e-01
E019   0.0691  1 7.93e-01
E020   0.7365  1 3.91e-01
E021   0.0780  1 7.80e-01
E022   0.4146  1 5.20e-01
E023   3.7310  1 5.34e-02
```

We see that there is one exon, `E010`, with a very small p value, while for all other exons, the p values are unremarkable.

A convenient interface which calls `testGeneForDEU` for all genes and fills the `pvalue` and `padjust` columns of the `featureData` slots of the *ExonCountSet* object with the results is provided by the function `testForDEU`.

```
> pasillaExons <- testForDEU(pasillaExons)
```

The function `DEUresultTable` provides a summary table of the results.

```
> pasillaExons <- estimatelog2FoldChanges(pasillaExons)
> res1 <- DEUresultTable(pasillaExons)
```

```
> table(res1$padjust < 0.1)

FALSE  TRUE 
  386     9 
```

```
> plot(res1$meanBase, res1[, "log2fold(untreated/treated)"], log = "x",
+     col = ifelse(res1$padjust < 0.1, "red", "black"), ylim = c(-4,
+         4), main = "pasilla MvsA")
```

4 Additional technical or experimental variables

In the previous section we performed the analysis of differential exon usage ignoring the information regarding the library type of the samples. In this case we introduce a technical variable, but the user can introduce an additional experimental variable.

```
> design(pasillaExons)

             condition replicate        type
treated1fb     treated         1 single-read
treated2fb     treated         2  paired-end
treated3fb     treated         3  paired-end
untreated1fb untreated         1 single-read
untreated2fb untreated         2 single-read
untreated3fb untreated         3  paired-end
untreated4fb untreated         4  paired-end
```

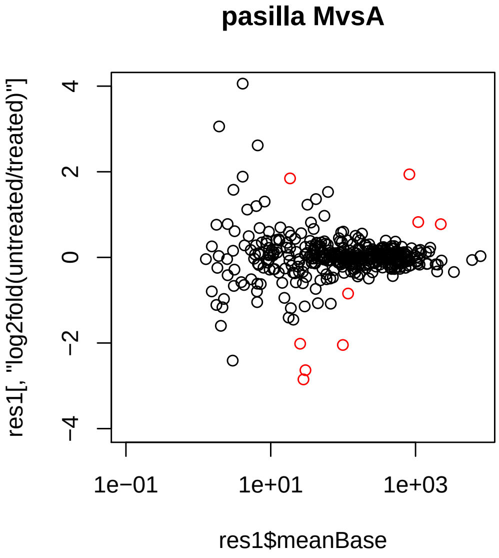

Figure 2: Mean expression vs log2 fold change plot, significant hits are colored in red.

In this section, we show how to take the factor `type` into account in the analysis. The objective of this is to avoid detecting the changes being introduced by this additional variable and not necessarily by the variable of interest, in this case `condition`. First, we need to provide the function `estimateDispersions` with a formula that makes it aware of the additional factor (besides `condition`, which it considers by default).

```
> formuladispersion <- count ~ sample + (exon + type) * condition
> pasillaExons <- estimateDispersions(pasillaExons, formula = formuladispersion)
> pasillaExons <- fitDispersionFunction(pasillaExons)
```

Second, for the testing, we will also change the two formulas to take into account the library type.

```
> formula0 <- count ~ sample + type * exon + condition
> formula1 <- count ~ sample + type * exon + condition * I(exon ==
+     exonID)
```

```
> pasillaExons <- testForDEU(pasillaExons, formula0 = formula0,
+     formula1 = formula1)
> res2 <- DEUresultTable(pasillaExons)
```

```
> table(res2$padjust < 0.1)
```

```
FALSE  TRUE
  384    11
```

```
> bottom = function(x) pmax(x, 1e-06)
> plot(bottom(res1$padjust), bottom(res2$padjust), log = "xy")
```

See Figure 3.

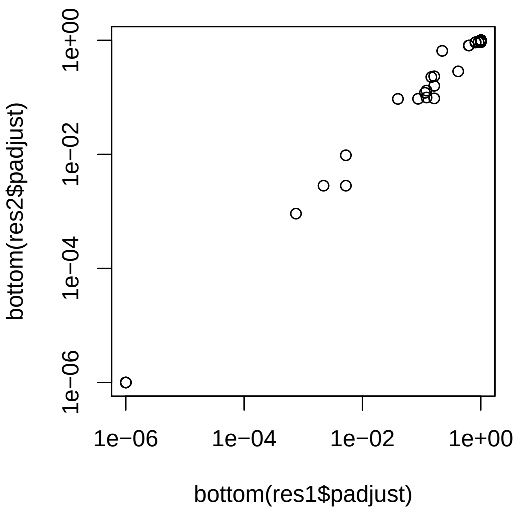

Figure 3: Comparison of differential exon usage p values from analysis with (y-axis, `res2`) and without (x-axis, `res1`) consideration of batch (library type) effects.

5 Visualization

DEXSeq has a function to visualize the results of `testForDEU`.

```
> plotDEXSeq(pasillaExons, "FBgn0010909", cex.axis = 1.2, cex = 1.3,
+     lwd = 2, legend = TRUE)
```

The result is shown in Figure 4. This plots shows the fitted expression values of each of the exons. Optionally, one can also visualize the transcript models (Figure 5), which might be useful for putting differential exon usage results into the context of isoform regulation.

```
> plotDEXSeq(pasillaExons, "FBgn0010909", displayTranscripts = TRUE,
+     cex.axis = 1.2, cex = 1.3, lwd = 2, legend = TRUE)
```

Other useful options are to look at the count values from the individual samples, rather than at the model effect estimates. For this display, it is useful to normalize the counts by the size factors (Figure 6).

```
> plotDEXSeq(pasillaExons, "FBgn0010909", expression = FALSE, norCounts = TRUE,
+     cex.axis = 1.2, cex = 1.3, lwd = 2, legend = TRUE)
```

Additionally, one can also visualize the fitted splicing, same as in Figure 4, but taking away the overall expression value of the gene.

```
> plotDEXSeq(pasillaExons, "FBgn0010909", cex.axis = 1.2, cex = 1.3,
+     lwd = 2, legend = TRUE, expression = FALSE, splicing = TRUE)
```

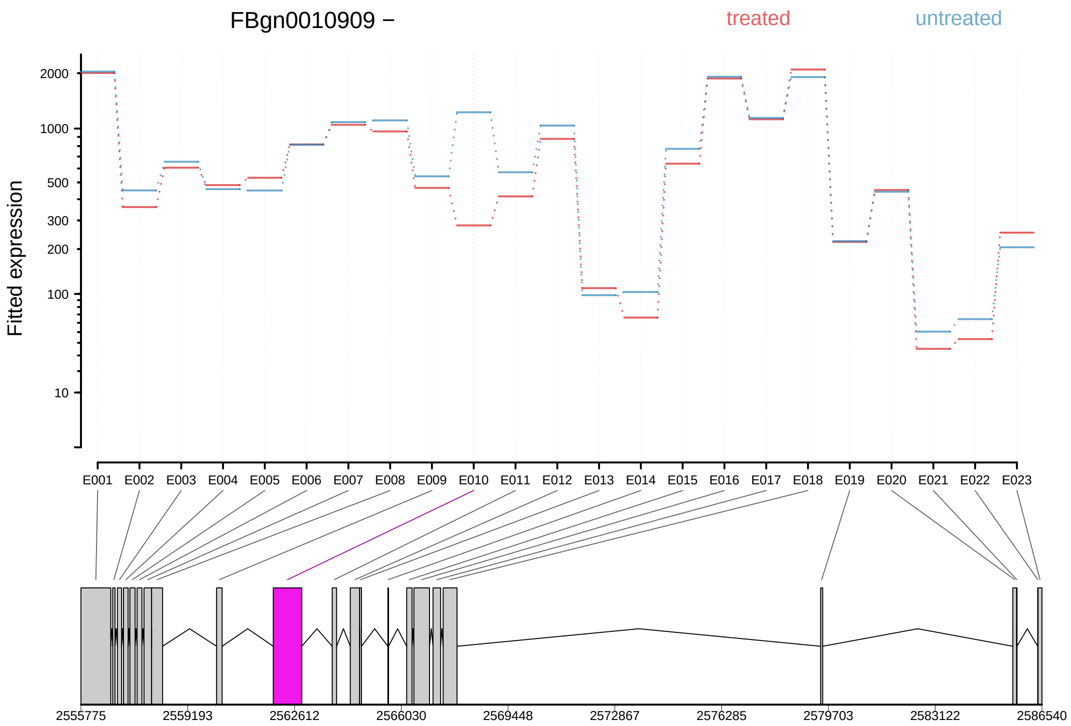

Figure 4: The plot represents the expression estimates from a call to `testForDEU`. Shown in red is the exon that showed significant differential exon usage.

To generate an easily browsable, detailed overview over all analysis results, the package provides an HTML report generator, implemented in the function *DEXSeqHTML*. This function uses the package *hwriter* to create a result table with links to plots for the significant results, allowing a more detailed exploration of the results. To see an example, visit `http://www.embl.de/~reyes/DEXSeqReport/testForDEU.html`. The report shown there was generated using this code.

```
> DEXSeqHTML(pasillaExons, FDR = 0.1, color = c("#FF000080", "#0000FF80"))
```

6 Parallelization

DEXSeq analysis can be computationally slow with large datasets due to the number of iterations and glms that are fitted, especially with datasets with a big number of samples, or organisms containing genes with a large number of exons. There are some steps of the analysis that require the whole dataset, but the two parts that are most time consuming can be parallelized (functions `estimateDispersions` and `testForDEU`) by changing the parameter nCores in the functions. Internally, this functions will subset the *ExonCountSet* object into smaller objects to distribute them to different cores. Please note that only the Cox-Reid calculation of the dispersion can be parallelized, the mean-variance dependent fit is done with all the dataset. This parameter depends on the *multicore* package, but the user can do the subsetting manually in order to parallelized with other packages, e.g. *Rmpi*.

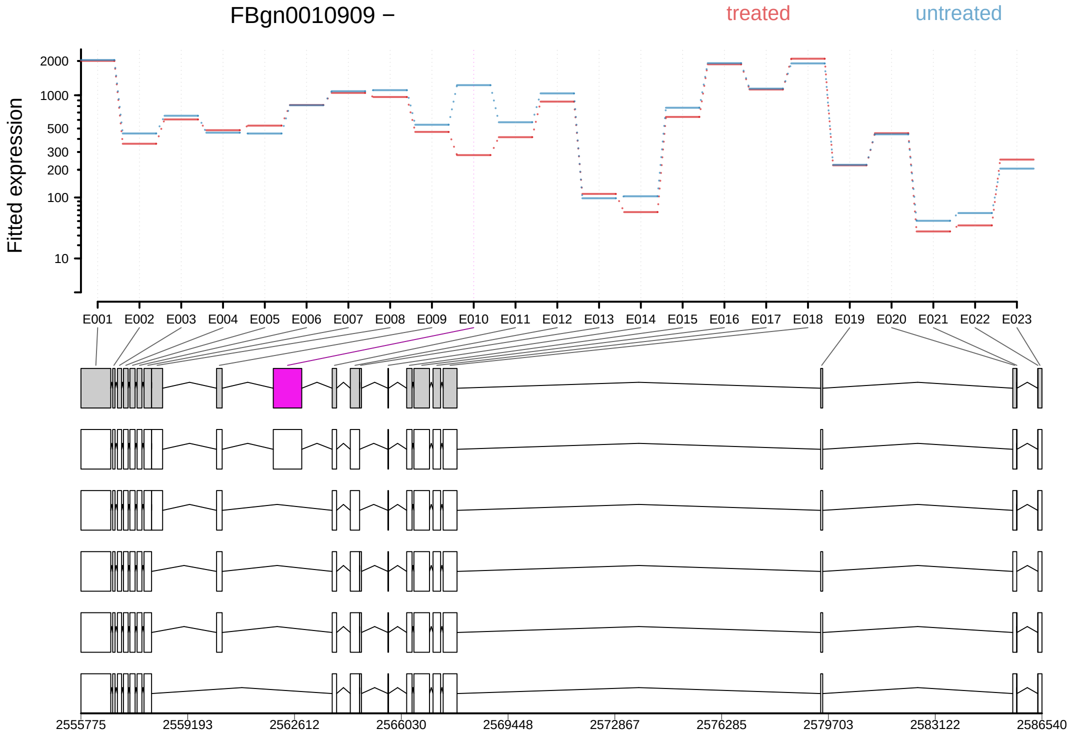

Figure 5: As in Figure 4, but including the annotated transcript models.

```
> data("pasillaExons", package = "pasilla")
> library(multicore)
> pasillaExons <- estimateSizeFactors(pasillaExons)
> pasillaExons <- estimateDispersions(pasillaExons, nCores = 3,
+     quiet = TRUE)
> pasillaExons <- fitDispersionFunction(pasillaExons)
> pasillaExons <- testForDEU(pasillaExons, nCores = 3)
```

7 Making a routinary differential exon usage analysis

In the previous sections, we pass through each of the steps to make an analysis for differential exon usage using *DEXSeq*. However, this is a very fragmented work flow, in case that the user wants to perform a routinary analysis (e.g. same analysis on different `ExonCountSet` objects) in which all the parameters are already known, the function `makeCompleteDEUAnalysis` can be useful. All the analysis we did before could be done by a single function call.

```
> data("pasillaExons", package = "pasilla")
> pasillaExons <- makeCompleteDEUAnalysis(pasillaExons, formulaDispersion = formuladispersion,
+     formula0 = formula0, formula1 = formula1, nCores = 1)
```

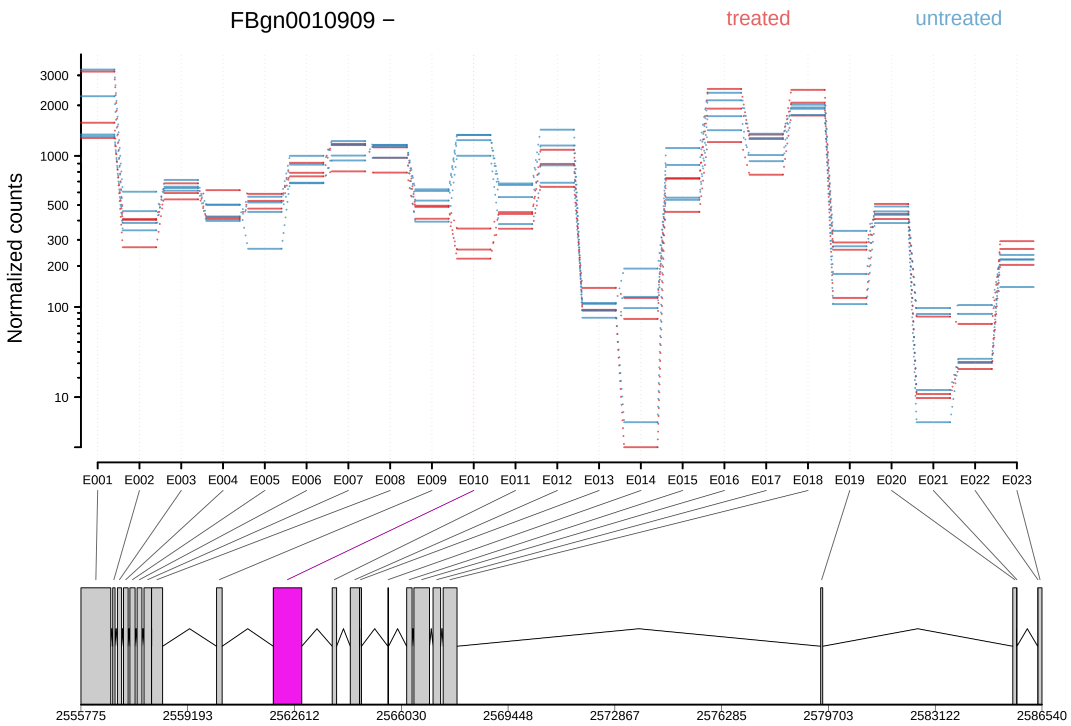

Figure 6: As in Figure 4, with normalized count values of each exon in each of the samples.

8 Creating *ExonCountSet* objects

8.1 From files produced by HTSeq

In this section, we describe how to create an *ExonCountSet* from an alignment of the RNA-seq reads to the genome, in SAM format, and a file describing gene and transcript models in GTF format.

The first steps of this workflow involve two scripts for the Python library *HTSeq*. These scripts are provided as part of the R package *DEXSeq*. The first script, `dexseq_prepare_annotation.py`, parses an annotation file in GTF format to define non-overlapping exonic parts: for instance, consider a gene whose transcripts contain either of two exons whose genomic regions overlap. In such a case, the script defines three exonic regions: two for the non-overlapping parts of each of the two exons, and a third one for the overlapping part. The script produces as output a new file in GTF format. The second script, `dexseq_count.py`, reads the GTF file produced by `dexseq_prepare_annotation.py` and an alignment in SAM format and counts the number of reads falling in each of the defined exonic parts.

The files that were used in this way to create the `pasillaGenes` object are provided within the *pasilla* package:

```
> dir(system.file("extdata", package = "pasilla"))
```

```
 [1] "Dmel.BDGP5.25.62.DEXSeq.chr.gff" "geneIDsinsubset.txt"
 [3] "pasilla_gene_counts.tsv"         "treated1fb.txt"
```

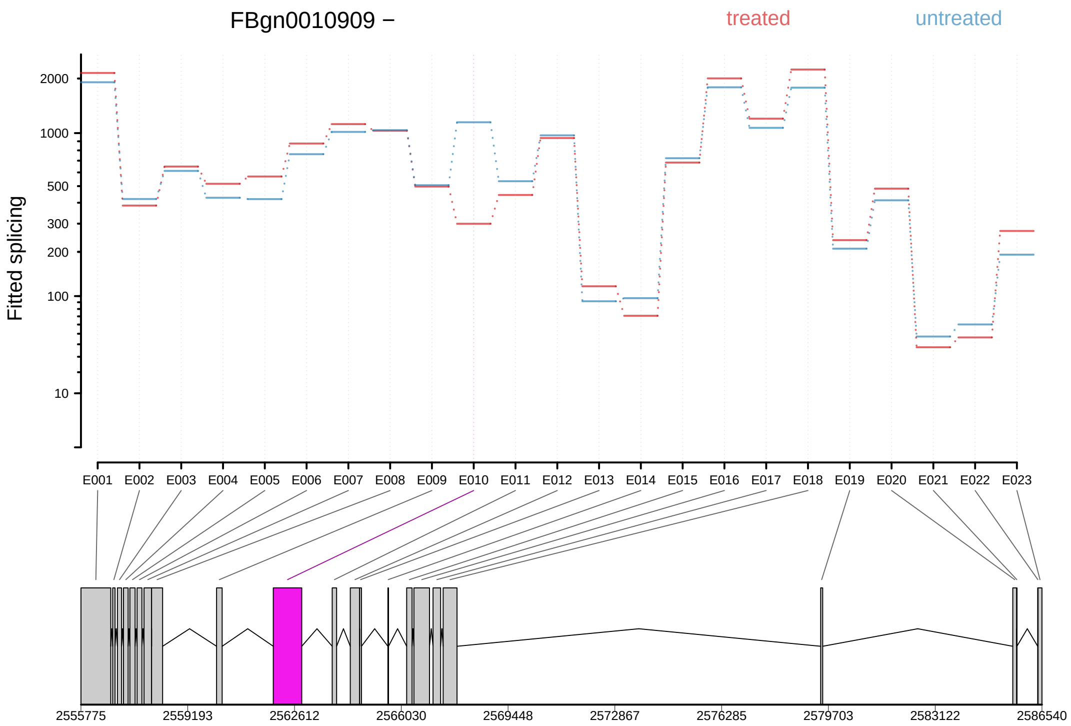

Figure 7: The plot represents the splicing estimates, as in Figure 4, but taking away the overall gene expression.

```
 [5] "treated2fb.txt"                   "treated3fb.txt"
 [7] "untreated1fb.txt"                 "untreated2fb.txt"
 [9] "untreated3fb.txt"                 "untreated4fb.txt"
```

The vignette[2] of the package *pasilla* provides a complete transcript of these steps.

The *DEXSeq* function `read.HTSeqCounts` is then able to read the output from these scripts and returns an *ExonCountSet* object with the relevant information for differential exon usage analysis and visualization.

8.2 From elementary R data structures

Users can also provide their own data, contained in elementary R objects, directly to the function `newExonCountSet` in order to create an *ExonCountSet* object. The package *GenomicRanges* in junction with the annotation packages available in *Bioconductor* give alternative options that allow users to create the objects necessary to make an *ExonCountSet* object using only R. The minimum requirements are

1. a per-exon count matrix, with one row for every exon and one column for every sample,

2. a vector, matrix or data frame with information about the samples, and

[2] Data preprocessing and creation of the data objects pasillaGenes and pasillaExons

3. two vectors of gene and exon identifiers that align with the rows of the count matrix.

With such a minimal object, it is possible to perform the analysis for differential exon usage, but the visualization functions will not be so useful. The necessary information about exons start and end positions can be given as a data frame to the `newExonCountSet` function, or can be added to the *ExonCountSet* object after its creation via the `featureData` accessor. For more information, please see the manual page of `newExonCountSet`.

```
> bare <- newExonCountSet(countData = counts(pasillaExons), design = design(pasillaExons),
+     geneIDs = geneIDs(pasillaExons), exonIDs = exonIDs(pasillaExons))
```

9 Gene count table

The function `geneCountTable` computes a table of *gene counts*, which are obtained by summing the counts from all exons with the same geneID. This might be useful for the detection of differential expression of genes, where the table can be used as input e. g. for the packages *DESeq* or *edgeR*. This kind of table can also be produced with the package *GenomicRanges*, e. g. function `summarizeOverlaps`.

```
> head(geneCountTable(pasillaExons))
```

```
            treated1fb treated2fb treated3fb untreated1fb untreated2fb
FBgn0000256       2285       2231       1889         2369         4675
FBgn0000578       6748       5808       5520         6910        11338
FBgn0002921      18397      19684      15691        15686        21674
FBgn0003089         12          8          7            3           16
FBgn0010226        212        248        228          184          252
FBgn0010280       4085       4611       4225         4145         6597
            untreated3fb untreated4fb
FBgn0000256         2191         2138
FBgn0000578         6042         5351
FBgn0002921        10953        11842
FBgn0003089           13            9
FBgn0010226          131          193
FBgn0010280         3914         3768
```

References

[1] Simon Anders and Wolfgang Huber. Differential expression analysis for sequence count data. *Genome Biology*, 11:R106, 2010.

[2] A. N. Brooks, L. Yang, M. O. Duff, K. D. Hansen, J. W. Park, S. Dudoit, S. E. Brenner, and B. R. Graveley. Conservation of an RNA regulatory map between Drosophila and mammals. *Genome Research*, pages 193–202, 2011.

[3] Mark D. Robinson and Gordon K. Smyth. Moderated statistical tests for assessing differences in tag abundance. *Bioinformatics*, 23(21):2881–2887, 2007.

10 Session Information

```
> sessionInfo()

R version 2.14.0 (2011-10-31)
Platform: x86_64-unknown-linux-gnu (64-bit)

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=C                 LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

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

other attached packages:
[1] DEXSeq_1.0.2   Biobase_2.14.0

loaded via a namespace (and not attached):
[1] hwriter_1.3    plyr_1.6       statmod_1.4.14 stringr_0.5    tools_2.14.0
```