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

Alejandro Reyes, Simon Anders, Wolfgang Huber

European Molecular Biology Laboratory (EMBL),
Heidelberg, Germany

DEXSeq version 1.6.0 (Last revision 2013-02-27)

Abstract

This vignette describes how to use the Bioconductor package *DEXSeq* to detect quantitatively different usage of exons from shotgun RNA sequence (RNA-seq) data. The statistical model is based on generalised linear models of the Negative Binomial family (NB-GLMs) and aims to detect changes between experimental conditions of interest that are significantly larger than the technical and biological variability among replicates. The method is described in [2]. It is a specialisation of the NB-GLM approach at the overall gene level provided by the *DESeq* [1] and *edgeR* [4] packages. As input, *DEXSeq* uses the number of reads mapping to each of the exons of a genome. Here, the method is demonstrated on the data from the package *pasilla*. To cite this software, please refer to `citation("DEXSeq")`.

Contents

1 The Pasilla data set

Brooks et al. [3] investigated the effect of siRNA knock-down of the gene *pasilla* on the transcriptome of fly S2-DRSC cells. The *pasilla* protein is known to bind to mRNA in the spliceosome and is thought to be involved in the regulation of splicing. The *pasilla* gene is the Drosophila melanogaster ortholog of mammalian NOVA1 and NOVA2. The data set, 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 of the untreated control.

Here, we will use these data to demonstrate the *DEXSeq* package. They are provided in the object `pasillaExons` in the *pasilla* package. We start by loading the *DEXSeq* package and the example data.

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

The `data` command above has loaded the object `pasillaExons`, which is an object of class *ExonCountSet*. This is the central data class of *DEXSeq*: at the beginning of an analysis, the user creates an *ExonCountSet* object that contains all the requried data and metadata. In the course of the analysis workflow, the intermediate and final results of computations are stored in the object, too.

We defer the discussion of how you can create such an object from your own data to Section 9 and instead start with inspecting the example data object.

The *ExonCountSet* class is derived from *eSet*, Bioconductor's standard container class for experimental data. As such, it contains the usual accessor functions for sample, feature and assay data (including `pData`, `fData`, `experimentData`), and some specific ones. The accessor function `design` shows the available sample annotations.

```
> design(pasillaExons)
```

```
             condition        type
treated1fb     treated single-read
treated2fb     treated  paired-end
treated3fb     treated  paired-end
untreated1fb untreated single-read
untreated2fb untreated single-read
untreated3fb untreated  paired-end
untreated4fb untreated  paired-end
```

The read counts can be accessed with the `counts` function. We print the first 20 rows of this table:

```
> head( counts(pasillaExons), 20 )
```

	treated1fb	treated2fb	treated3fb	untreated1fb	untreated2fb	untreated3fb	untreated4fb
FBgn0000256:E001	92	28	43	54	131	51	49
FBgn0000256:E002	124	80	91	76	224	82	95
FBgn0000256:E003	340	241	262	347	670	260	297
FBgn0000256:E004	250	189	201	219	507	242	250
FBgn0000256:E005	96	38	39	71	76	57	62
FBgn0000256:E006	1	0	1	0	2	0	2
FBgn0000256:E007	149	70	71	130	281	115	94
FBgn0000256:E008	190	124	129	137	345	137	136
FBgn0000256:E009	43	17	25	4	9	2	4
FBgn0000256:E010	26	13	11	8	20	6	5
FBgn0000256:E011	14	3	8	4	20	3	2
FBgn0000256:E012	1	2	2	0	1	0	0
FBgn0000256:E013	2	2	4	0	0	0	0
FBgn0000256:E014	2	0	1	0	0	0	0
FBgn0000256:E015	58	28	44	46	131	96	69
FBgn0000256:E016	93	22	34	73	209	54	36
FBgn0000256:E017	1	0	0	0	0	0	0
FBgn0000578:E001	2	0	0	1	0	0	1
FBgn0000578:E002	3	4	0	2	0	0	1
FBgn0000578:E003	96	34	27	74	64	18	26

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

The rows are labelled with gene IDs (here Flybase IDs), followed by a colon and a *counting bin* number. A counting bin corresponds to an exon or part of an exon. The table content indicates the number of reads that have been mapped to each counting bin in the respective sample.

To see details on the counting bin, 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:E001 FBgn0000256   E001 chr2L 3872658 3872947      -
FBgn0000256:E002 FBgn0000256   E002 chr2L 3873019 3873322      -
FBgn0000256:E003 FBgn0000256   E003 chr2L 3873385 3874395      -
FBgn0000256:E004 FBgn0000256   E004 chr2L 3874450 3875302      -
FBgn0000256:E005 FBgn0000256   E005 chr2L 3878895 3879067      -
FBgn0000256:E006 FBgn0000256   E006 chr2L 3879652 3880038      -
```

`pasillaExons` contains only a subset of 46 genes that we selected from the genome-wide data set of [3]; we consider this subset so that this vignette can be run quickly. For your own analyses, you would typically consider a genome-wide data set. Of the 46 genes, there is one with 36 exons, and three with 16 exons:

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

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

In Section 9, we explain how you can create analogous data objects from your own data.

2 Normalisation

Different samples might be sequenced with different depths. In order to adjust for such coverage biases, we estimate *size factors*, which measure relative sequencing depth. *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 untreated4fb
       1.336        0.800        0.922        0.991        1.568        0.838        0.830
```

2.1 Helper functions

The `.Rnw` file that underlies this document contains the definition of helper functions that this lab uses for the preparation of plots: `plotDispEsts` and `plotMA`. Before preceding, please run the code chunk where these functions are defined, so you have them available in your workspace. (In future versions of the *DEXSeq* package, these functions will already be defined in the package.)

3 Dispersion estimation

To test for differential expression, we need to estimate the variance of the data. This is necessary to be able to distinguish technical and biological variation (noise) from real effects on exon expression due to the different conditions. The information on the size of the noise is infered from the biological replicates in the data set. However, in RNA-seq experiments the number of replicates is often 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.

The first step is to get a dispersion estimate for each exon. This task is performed by the function *estimateDispersions*, using Cox-Reid (CR) likelihood estimation (our method follows that of the package *edgeR* [4]). Before starting estimating the CR dispersion estimates, `estimateDispersions` first defines the "testable" exons, which fulfill the following criteria:

1. The exon's total sum of counts over all samples is higher than the parameter `minCount`,
2. the exon comes from a gene that has at most `maxExon` exons, and

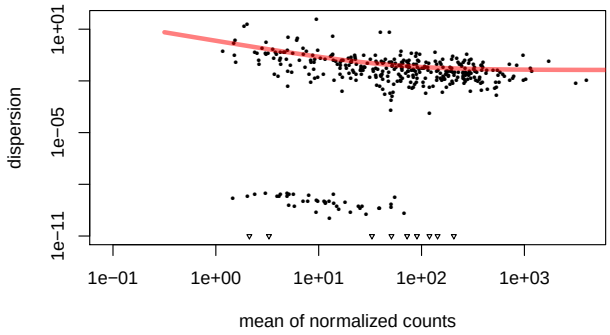

Figure 1: Per-gene dispersion estimates (shown by points) and the fitted mean-dispersion function (red line).

3. the exon comes from a gene that has more than one "testable" exon.

These testable exons are marked in the column `testable` of the feature data. Then, a CR estimate is computed for each gene, and the obtained values are stored in the feature data column *dispBeforeSharing*.

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

Note that for a full, genome-wide data set, execution of this function may take a while. To indicate progress, one dot is printed on the console whenever 100 genes have been processed. If you have a machine with multiple cores, you may want to use the `nCores` option to instruct the function to parallelize the task over several CPU cores. See Section 7 and the function's help page for details.

Afterwards, the function `fitDispersionFunction` needs to be called, in which a dispersion-mean relation $\alpha(\mu) = \alpha_0 + \alpha_1/\mu$ is fitted to the individual CR dispersion values (as stored in *dispBeforeSharing*). The fit 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.[2] See our paper [2] for the rationale behind this "sharing" scheme.

```
> pasillaExons <- fitDispersionFunction(pasillaExons)

> head(fData(pasillaExons)$dispBeforeSharing)

[1] 0.00930 0.00826 0.01669 0.01912 0.07736      NA

> pasillaExons@dispFitCoefs

     (Intercept) I(1/means[good])
          0.0429           2.0872

> head(fData(pasillaExons)$dispFitted)

[1] 0.0787 0.0631 0.0493 0.0511 0.0773 2.6909
```

As a fit diagnostic, we plot the per-exon dispersion estimates versus the mean normalised count.

```
> plotDispEsts( pasillaExons )
```

The plot (Figure 1) shows the estimates for all exons as dots and the fit as red line. The red line follows the trend of the dots in the upper cluster of dots. The lower cluster stems from exons for which the sample noise happens to fall below shot noise, i. e., their sample estimates of the dispersion is zero or close to zero and hence form another cluster at the bottom. The fact that these two clusters look so well separated is to a large extent an artifact of the logarithmic y-axis scaling. Inspect the fit and make sure that the regression line follows the trend of the points within the upper cluster.

In Section 5, we will see how to incorporate further experimental or technical variables into the dispersion estimation.

[2] Especially when the dispersion estimates are very large, this fit can be difficult, and has occasionally caused the function to fail. In these rare cases please contact the developers.

4 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)
```
(1)

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

```
sample + exon + condition.
```
(2)

The deviances of both fits are compared using a χ^2-distribution. Based on that, we will be able to decide whether the null model (2) is sufficient to explain the data, or whether it can be rejected in favour of the alternative, model (1).

The function `testForDEU` performs such a test, one after another, for each exon in each gene and fills the `pvalue` and `padjust` columns of the `featureData` slots of the *ExonCountSet* object with the results. Here, `pvalue` contains the p values from the χ^2 test, and`padjust` is the result of performing Benjmini-Hochberg adjustment for mutliple testing on Robjectpvalue.

```
> pasillaExons <- testForDEU( pasillaExons )

> head( fData(pasillaExons)[, c( "pvalue", "padjust" ) ] )
```

```
                 pvalue padjust
FBgn0000256:E001  0.851   0.999
FBgn0000256:E002  0.560   0.999
FBgn0000256:E003  0.768   0.999
FBgn0000256:E004  0.680   0.999
FBgn0000256:E005  0.943   0.999
FBgn0000256:E006     NA      NA
```

For some usages, we may also want to estimate fold changes. To this end, we call `estimatelog2FoldChanges`:

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

Now, we can get a table of all results with `DEUresultTable`.

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

```
                     geneID exonID dispersion pvalue padjust meanBase
FBgn0000256:E001 FBgn0000256   E001     0.0787  0.851   0.999   58.343
FBgn0000256:E002 FBgn0000256   E002     0.0631  0.560   0.999  103.333
FBgn0000256:E003 FBgn0000256   E003     0.0493  0.768   0.999  326.476
FBgn0000256:E004 FBgn0000256   E004     0.0511  0.680   0.999  253.654
FBgn0000256:E005 FBgn0000256   E005     0.0774  0.943   0.999   60.638
FBgn0000256:E006 FBgn0000256   E006     2.6909     NA      NA    0.788
                 log2fold(untreated/treated)
FBgn0000256:E001                      0.0235
FBgn0000256:E002                     -0.0522
FBgn0000256:E003                      0.0226
FBgn0000256:E004                      0.0364
FBgn0000256:E005                     -0.0130
FBgn0000256:E006                      0.1290
```

Controlling false discovery rate (FDR) at 0.1 (10%), we can now ask how many counting bins show evidence of differential exon usage:

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

```
FALSE  TRUE
  376     7
```

We may also ask how many genes are affected

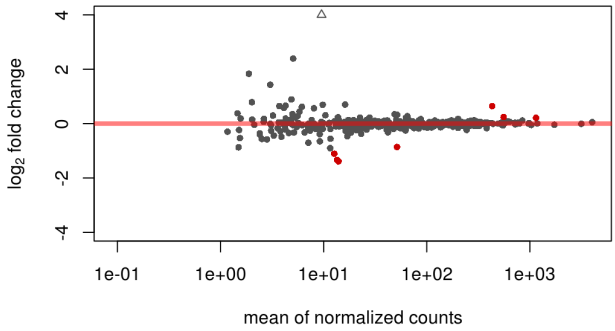

Figure 2: Mean expression versus $\log_2$ fold change plot. Significant hits (at `padj`<0.1) are colored in red.

```
> table ( tapply( res1$padjust < 0.1, geneIDs(pasillaExons), any ) )
```

```
FALSE  TRUE
   16     6
```

Remember that out example data set contains only a selection of genes. We have chosen these to contain interesting cases; so this large fraction of affected genes is not typical.

To see how the power to detect differential exon usage depends on the number of reads that map to an exon, a so-called MA plot is useful, which plots the logarithm of fold change versus average normalized count per exon and marks by red colour the exons with adjusted p values of less than 0.1 (Figure 2). There is of course nothing special about the number 0.1, and you can specify other thresholds in the call to `plotMA`. For the definition of the function `plotMA`, see Section 2.1.

```
> plotMA(with(res1, data.frame(baseMean = meanBase,
+                              log2FoldChange = `log2fold(untreated/treated)`,
+                              padj = padjust)),
+      ylim=c(-4,4), cex=0.8)
```

5 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 section, we show how to introduce this additional variable into the analysis. In this case, `type` is a technical variable, but additional experimental variables can be introduced in the same manner.

```
> design(pasillaExons)
```

```
             condition        type
treated1fb     treated single-read
treated2fb     treated  paired-end
treated3fb     treated  paired-end
untreated1fb untreated single-read
untreated2fb untreated single-read
untreated3fb untreated  paired-end
untreated4fb untreated  paired-end
```

First, we need to provide the function `estimateDispersions` with a model formula that makes it aware of the additional factor. Note that if the function `estimateDispersions` is called with no value for its `formula` argument (as we did in Section 3), the factor `condition` is considered by default.

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

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

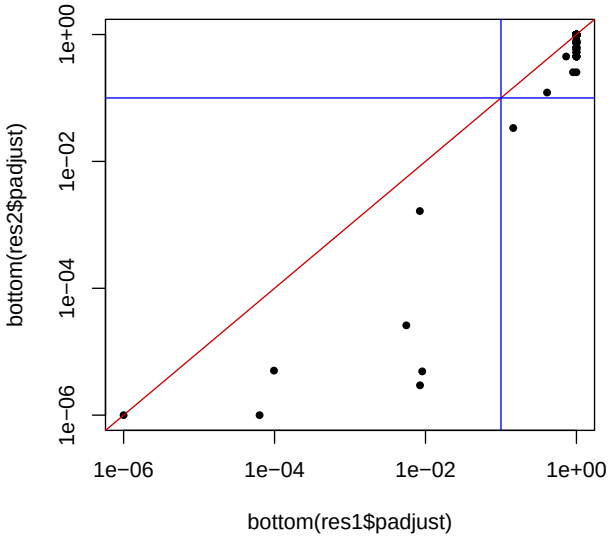

Figure 3: Comparison of differential exon usage p values from analysis with (y-axis, `res2`) and without (x-axis, `res1`) consideration of batch (library type) effects.

```
> 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 )
```

We can now compare with the previous result:

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

FALSE  TRUE
  375     8

> table(res1$padjust < 0.1, res2$padjust < 0.1)

        FALSE TRUE
  FALSE   375    1
  TRUE      0    7

> bottom = function(x) pmax(x, 1e-6)
> plot(bottom(res1$padjust), bottom(res2$padjust), log="xy", pch=20)
> abline(a=0,b=1, col="red3")
> abline(v=0.1, h=0.1, col="blue")
```

We can see from Figure 3 and the table above that with the *type*-aware analysis, detection power for differential exon usage due to *condition* is improved.

6 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 plot shows the fitted expression values of each of the exons. The function `plotDEXSeq` takes at least two arguments, the `pasillaExons` object and the gene ID. The option `legend=TRUE` causes a legend to be included. The three remaining arguments in the code chunk above are ordinary plotting parameters

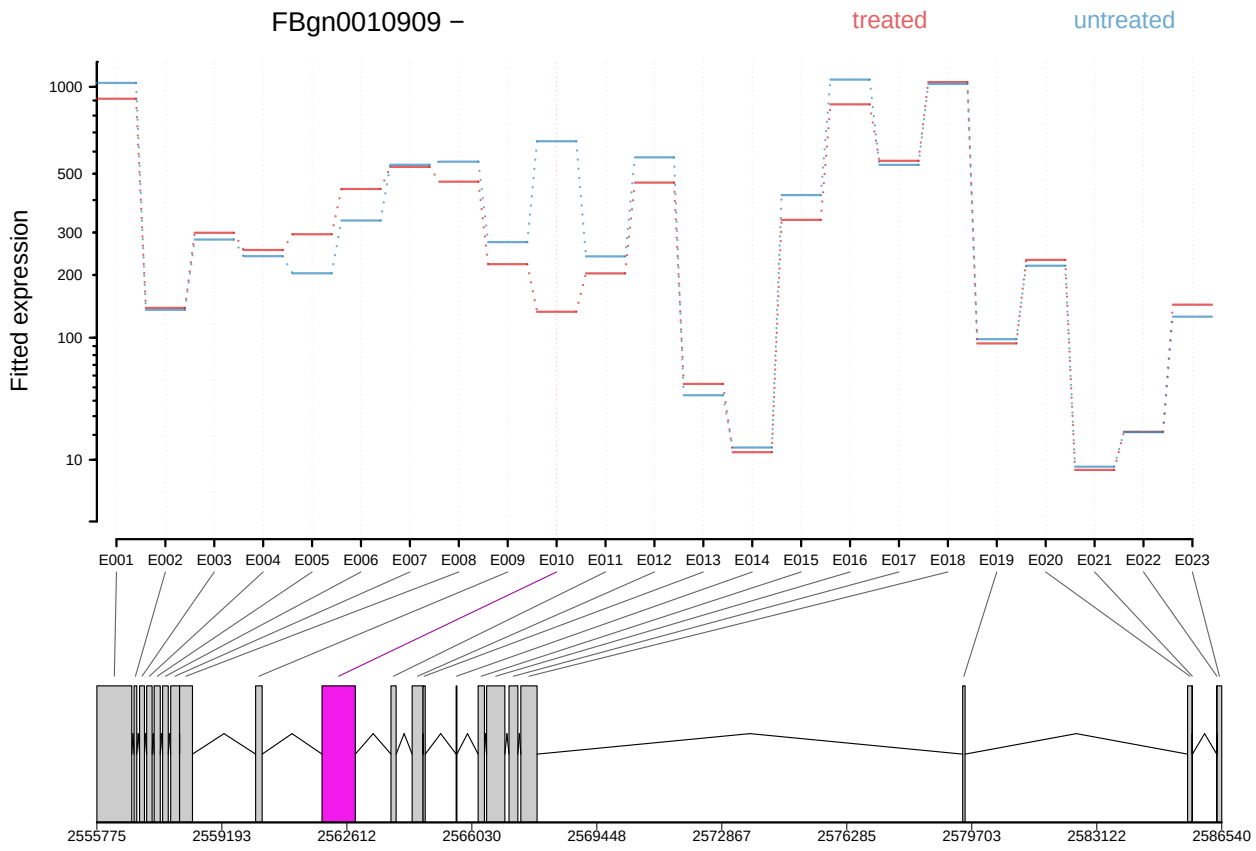

Figure 4: The plot represents the expression estimates from a call to `testForDEU`. Shown in red is the exon that showed significant differential exon usage.

which are simply handed over to R's standard plotting functions. They are usually not needed and included here only to improve appearance of the plots for this vignette. See the help page for `par` for details.

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 (option `norCounts=TRUE`), the counts are normalized by dividing them by the size factors (Figure 6).

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

As explaoined in detail in the paper, *DEXSeq* is designed to find difference in exon usage, i.e. differences in expression that only some but not all exons show, while differences in the overall expression of a gene but not in the isoform abundance ratios (and hence affecting all exons in the same way) will not be considered evidence of diffential exon usage. Hence, it may be advantegeous to remove overall differences in expression also from the plots. Use the option `splicing=TRUE` for this purpose.

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

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 the code:

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

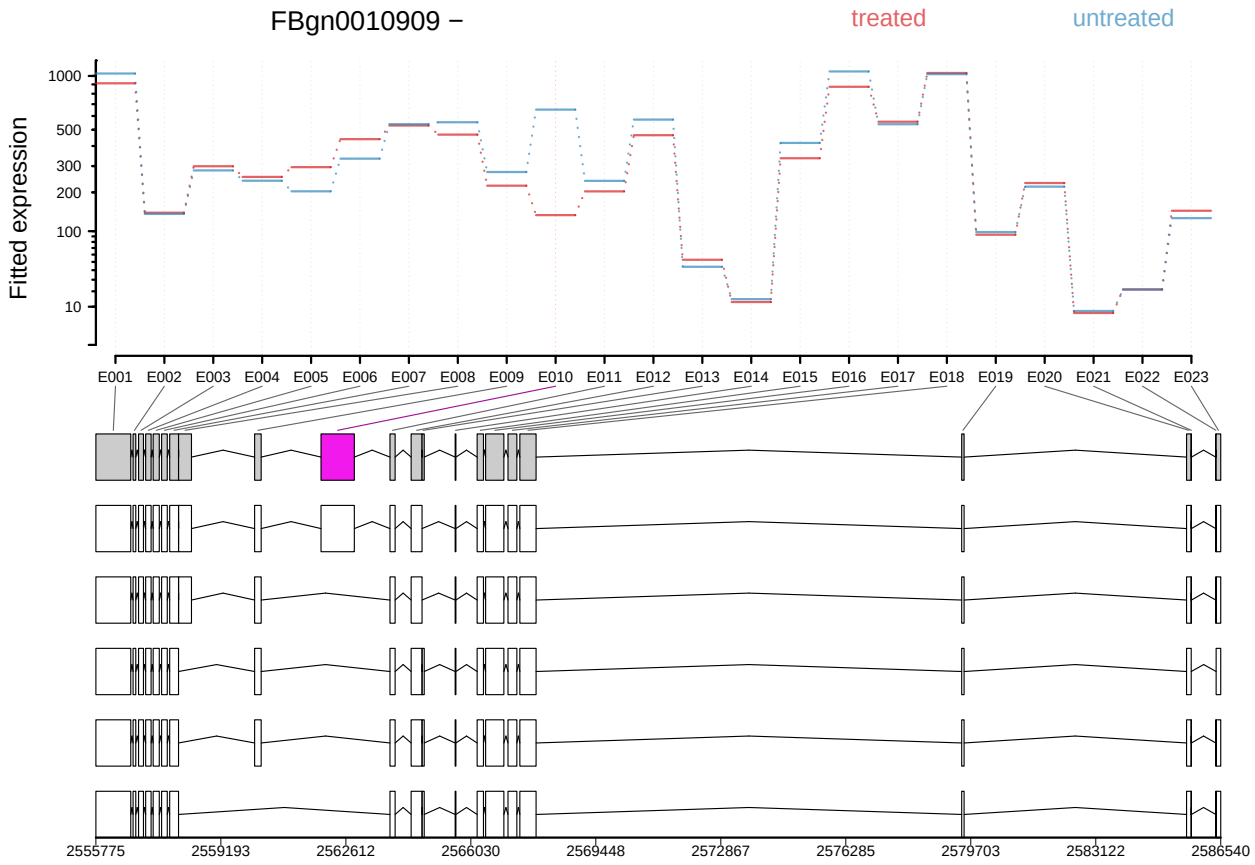

Figure 5: As in Figure 4, but including the annotated transcript models.

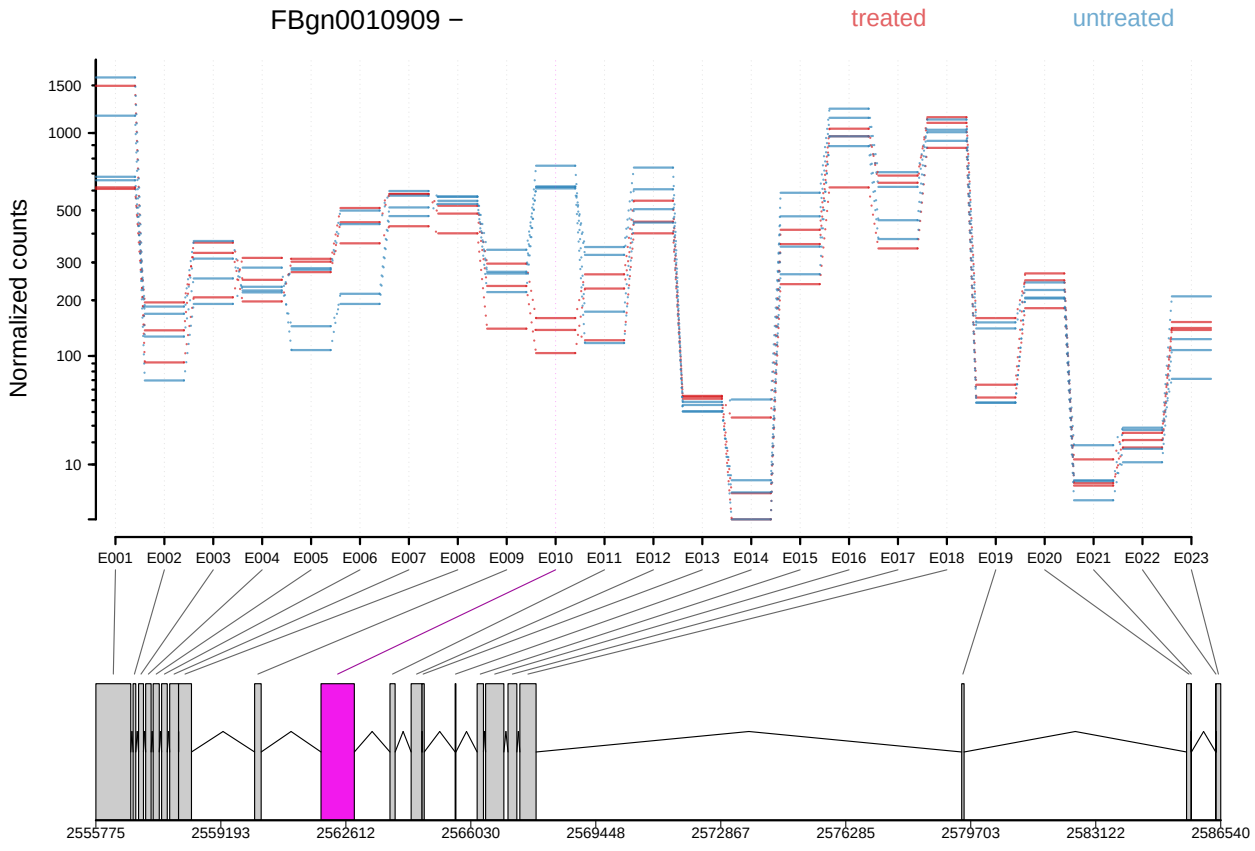

Figure 6: As in Figure 4, with normalized count values of each exon in each of the samples.

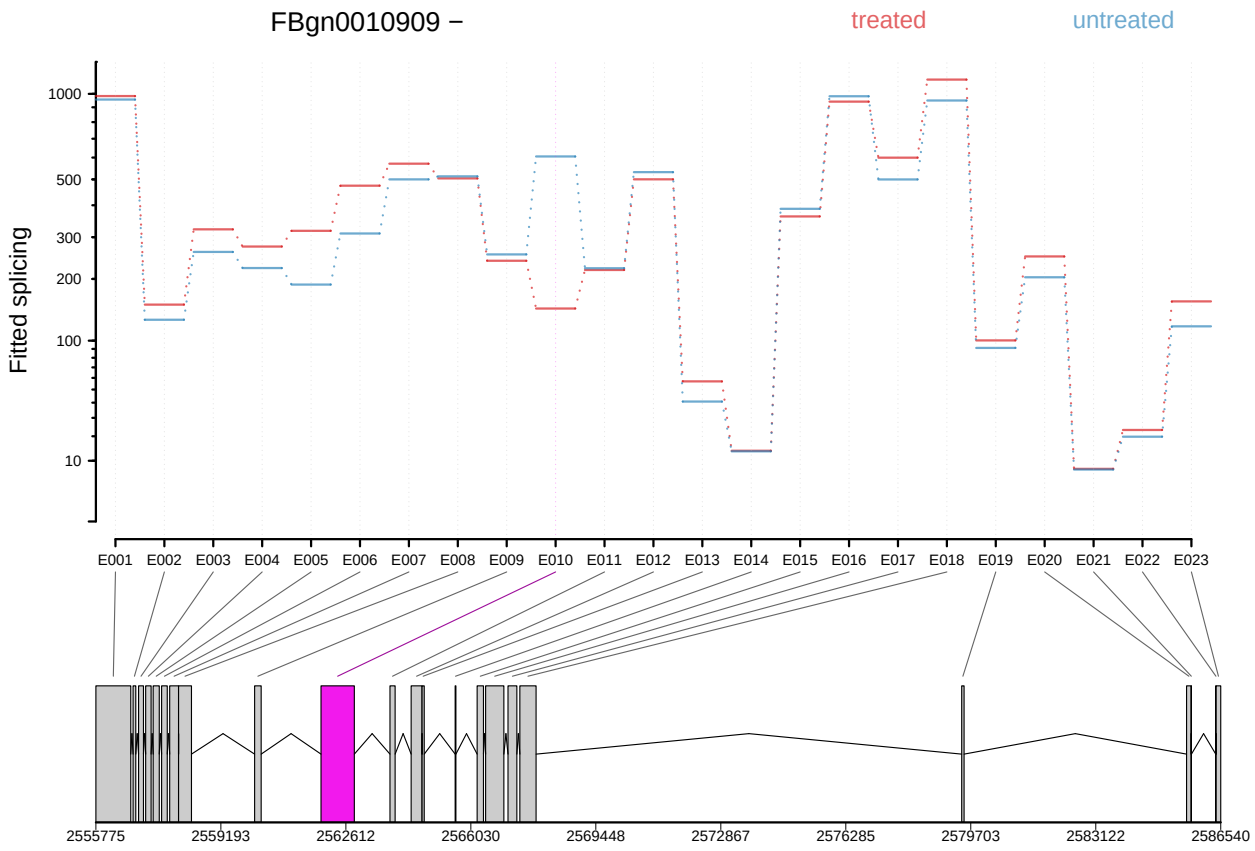

Figure 7: The plot represents the splicing estimates, as in Figure 4, but taking away the overall gene expression.

7 Parallelization

DEXSeq analyses can be computationally heavy, especially with data sets that comprise a large number of samples, or with genomes containing genes with large numbers of exons. While some steps of the analysis work on the whole data set, the two parts that are most time consuming (the functions `estimateDispersions` and `testForDEU`) can be parallelized by setting the parameter nCores. These functions will then distribute the *ExonCountSet* object into smaller objects that are processed in parallel on different cores. This functionality uses the *parallel* package.

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

8 Perform a standard differential exon usage analysis in one command

In the previous sections, we went through the analysis step by step. Once you are sufficiently confident about the work flow for your data, its invocation can be streamlined by the wrapper function `makeCompleteDEUAnalysis`, which runs the analysis shown above through a single function call.

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

9 Creating ExonCountSet objects

9.1 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*, and are installed in the sub-directory textttpython_scripts of the package's installation directory. To find the latter, use `system.file`.

```
> pkgDir = system.file(package="DEXSeq")
> pkgDir

[1] "/tmp/RtmpQWiG6E/Rinst26e0664dd30c/DEXSeq"

> list.files(pkgDir)

 [1] "CITATION"       "DESCRIPTION"    "INDEX"          "Meta"           "NAMESPACE"
 [6] "NEWS"           "R"              "doc"            "help"           "html"
[11] "python_scripts"

> list.files(file.path(pkgDir, "python_scripts"))

[1] "dexseq_count.py"              "dexseq_prepare_annotation.py"
```

To use the scripts, you need to first install *HTSeq* as explained on the web site [**?**]. Run the scripts from the command line. Both scripts display usage instructions when called without arguments.

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

The vignette[3] 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.

9.2 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
3. two vectors of gene and exon identifiers that align with the rows of the count matrix.

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

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

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`.

9.3 From GRanges, BamListFiles and transcriptDb objects

Alternatively, users can create their ExonCountSet objects from other *Bioconductor* data objects. The code for implementating these functions was kindly contributed by Mike Love. For details, check the *parathyroidSE* package. The workflow is the same as with the *HTSeq* python scripts. First we prepare the annotation file, in order to define non-overlapping exonic parts. In order to control the exons that are used by many genes, the parameter *aggregateGenes* allows the users to either ignore these exons and treat the genes separately, or merge the genes into a single "aggregate gene" and take into account these exons.

```
>   library(GenomicFeatures)
>   hse <- makeTranscriptDbFromBiomart(biomart="ensembl", dataset="hsapiens_gene_ensembl")
>   exonicParts <- prepareAnnotationForDEXSeq( hse, aggregateGenes=TRUE )
```

The `exonicParts` object contains a GRanges object, that we can further use to count the number of read fragments that overlap with our exonic bins. We do this using the function `countReadsForDEXSeq`.

```
>   bamDir <- system.file("extdata",package="parathyroidSE",mustWork=TRUE)
>   fls <- list.files(bamDir, pattern="bam$",full=TRUE)
>   bamlst <- BamFileList(fls)
>   SE <- countReadsForDEXSeq( exonicParts, bamlst )
```

Using the output of the functions `prepareAnnotationForDEXSeq` and `countReadsForDEXSeq` , we can call the function `buildExonCountSet` in order to build our typical `ExonCountSet` object.

```
>   ecs <- buildExonCountSet( SE, c("A", "A", "B"), exonicParts )
```

10 Lower-level functions

The following functions are not needed in the standard analysis work-flow, but may be useful for special purposes.

10.1 Single-gene functions

While the function `counts` returns the whole read count table, the function `countTableForGene` returns the count table for a single gene:

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

	treated1fb	treated2fb	treated3fb	untreated1fb	untreated2fb	untreated3fb	untreated4fb
E001	1997	494	562	1150	2514	570	547
E002	122	112	180	69	203	156	142
E003	276	293	305	190	398	312	259
E004	420	200	182	230	446	183	185
E005	416	217	279	146	170	237	231
E006	486	357	471	190	337	418	364

Both function `counts` and function `countTableForGene` can also return normalized counts (i.e., counts divided by the size factors). Use the option `normalized=TRUE`:

```
> head(countTableForGene(pasillaExons,"FBgn0010909", normalized=TRUE))
```

```
     treated1fb treated2fb treated3fb untreated1fb untreated2fb untreated3fb untreated4fb
E001     1494.8        618        609       1160.6         1603          680          659
E002       91.3        140        195         69.6          129          186          171
E003      206.6        366        331        191.7          254          372          312
E004      314.4        250        197        232.1          284          218          223
E005      311.4        271        302        147.3          108          283          278
E006      363.8        446        511        191.7          215          499          439
```

The function `modelFrameForGene` returns a model frame that can be used to fit a GLM for a single gene.

```
> mf <- modelFrameForGene( pasillaExons, "FBgn0010909" )
> head( mf )
```

```
      sample exon sizeFactor condition        type dispersion count
1 treated1fb E001       1.34   treated single-read     0.0166  1997
2 treated1fb E002       1.34   treated single-read     0.0562   122
3 treated1fb E003       1.34   treated single-read     0.0208   276
4 treated1fb E004       1.34   treated single-read     0.0221   420
5 treated1fb E005       1.34   treated single-read     0.0945   416
6 treated1fb E006       1.34   treated single-read     0.0405   486
```

This model frame can then be used, e.g., to estimate dispersions:

```
> mf <- estimateExonDispersionsForModelFrame( modelFrameForGene( pasillaExons, "FBgn0010909" ) )
```

Internally, the function `estimateDispersions` calls these two functions for each gene. It also stores the model frames for later use by `testForDEU`.

A single-gene version of `testForDEU` is also provided, by `testGeneForDEU`:

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

```
     deviance df  pvalue
E001 1.24e-01  1 0.72447
E002 7.09e-01  1 0.39981
E003 2.79e+00  1 0.09512
E004 2.90e+00  1 0.08867
E005 5.13e+00  1 0.02353
E006 7.61e+00  1 0.00581
E007 1.14e+00  1 0.28482
E008 1.57e-01  1 0.69153
E009 4.97e-01  1 0.48091
E010 1.47e+02  1 0.00000
E011 5.50e-02  1 0.81463
E012 4.77e-01  1 0.48985
E013 2.38e+00  1 0.12298
E014 1.47e-07  1 0.99969
E015 5.36e-01  1 0.46389
E016 5.33e-01  1 0.46516
E017 2.36e+00  1 0.12463
E018 1.99e+00  1 0.15851
E019 9.57e-04  1 0.97532
E020 2.03e+00  1 0.15441
E021 4.39e-03  1 0.94716
E022 2.17e-01  1 0.64119
E023 7.30e-01  1 0.39288
```

See the help pages of these function for further options (e. g., to specify formulae).

10.2 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 untreated3fb
FBgn0000256       1482        857        966          1169          2626          1105
FBgn0000578       4386       2301       2827          3541          6381          3139
FBgn0002921      11305       7135       8001          7433         11980          5618
FBgn0003089          8          4          4             6             9             4
FBgn0010226        129        100        113           106           126            60
FBgn0010280       2693       1776       2187          2088          3963          2069
           untreated4fb
FBgn0000256         1101
FBgn0000578         2725
FBgn0002921         5991
FBgn0003089            6
FBgn0010226           99
FBgn0010280         1981
```

Note that a read that mapped to several exons of a gene is counted for each of these exons by the `dexseq_count.py` script. The table returned `geneCountTable` will hence cout the read several time for the gene, which may skew downstream analyses in subtle ways. Hence, we recommend to use `geneCountTable` with care and use more sophisticated counting schemes where appropriate.

10.3 Further accessors

The function `geneIDs` returns the gene ID column of the feature data as a character vector and the function `exonIDs` return the exon ID column as a factor.

```
> head(geneIDs(pasillaExons))
```

```
FBgn0000256:E001 FBgn0000256:E002 FBgn0000256:E003 FBgn0000256:E004 FBgn0000256:E005
     FBgn0000256      FBgn0000256      FBgn0000256      FBgn0000256      FBgn0000256
FBgn0000256:E006
     FBgn0000256
14470 Levels: FBgn0000003 FBgn0000008 FBgn0000014 FBgn0000015 FBgn0000017 ... FBgn0261575
```

```
> head(exonIDs(pasillaExons))
```

```
FBgn0000256:E001 FBgn0000256:E002 FBgn0000256:E003 FBgn0000256:E004 FBgn0000256:E005
          "E001"           "E002"           "E003"           "E004"           "E005"
FBgn0000256:E006
          "E006"
```

These functions are useful for subsetting an *ExonCountSet* object.

References

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

[2] Simon Anders, Alejandro Reyes, and Wolfgang Huber. Detecting differential usage of exons from RNA-seq data. *Genome Research*, 22(10):2008–2017, 2012.

[3] 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.

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

11 Session Information

```
> sessionInfo()

R version 3.0.0 (2013-04-03)
Platform: x86_64-unknown-linux-gnu (64-bit)

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

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

other attached packages:
[1] pasilla_0.2.15     DESeq_1.12.0       lattice_0.20-15    locfit_1.5-9
[5] DEXSeq_1.6.0       Biobase_2.20.0     BiocGenerics_0.6.0

loaded via a namespace (and not attached):
 [1] AnnotationDbi_1.22.0 Biostrings_2.28.0    DBI_0.2-5            GenomicRanges_1.12.0
 [5] IRanges_1.18.0       RColorBrewer_1.0-5   RCurl_1.95-4.1       RSQLite_0.11.2
 [9] Rsamtools_1.12.0     XML_3.96-1.1         annotate_1.38.0      biomaRt_2.16.0
[13] bitops_1.0-5         genefilter_1.42.0    geneplotter_1.38.0   grid_3.0.0
[17] hwriter_1.3          splines_3.0.0        statmod_1.4.17       stats4_3.0.0
[21] stringr_0.6.2        survival_2.37-4      tools_3.0.0          xtable_1.7-1
[25] zlibbioc_1.6.0
```