easyRNASeq, an overview

Nicolas Delhomme

December 17, 2015

Contents

1 Before you start **2**

2 Changes to the vignette **2**

3 Introduction **4**

4 easyRNASeq **6**
- 4.1 usage . . . 6
- 4.2 warnings . . . 7
- 4.3 arguments . . . 8
- 4.4 Critical information regarding the input . . . 8
- 4.5 different input . . . 11
- 4.6 different annotation . . . 12
- 4.7 different output . . . 15
- 4.8 different summarization . . . 16
- 4.9 optional correction or normalization . . . 17
- 4.10 Performance . . . 19

5 Advanced usage **20**
- 5.1 De-multiplexing samples . . . 20

6 Yet to come **24**

7 Use Cases **25**
- 7.1 Creating a set of synthetic transcripts . . . 25
- 7.2 Processing a set of human samples . . . 27
- 7.3 Dealing with annotation inconsistencies . . . 31

8 FAQ **35**
- 8.1 Does easyRNASeq support reads of variable length? . . . 35
- 8.2 Does the easyRNASeq support paired-end reads? . . . 35

9 Session Information **36**

10 Final remarks **37**

1 Before you start

If you are completely new to the R/Bioconductor [1] packages dealing with Next-Generation Sequencing, you might want to start by doing the tutorial available in the `RnaSeqTutorial` vignette in the data package *RnaSqTutorial*. The same holds true if you're interested in understanding the details behind the content of the present vignette. If neither nor, just go ahead.

Moreover, if you find *easyRNASeq* useful and apply it in the frame of your research for a publication, please cite it:

easyRNASeq: a bioconductor package for processing RNA-Seq data
Nicolas Delhomme; Ismael Padioleau; Eileen E. Furlong; Lars M. Steinmetz
Bioinformatics 2012; doi: 10.1093/bioinformatics/bts477

2 Changes to the vignette

The last changes are reported in red.

1.8.4 Converted an existing paragraph on annotation warning(s) into a section to add additional details - see section 4.2 (page 7). Extended the first use case on synthetic transcript generation - see section 7.1 (page 25) - to introduce how to check for annotation overlaps. Added another section detailing input file characteristics and their impact on counting - see section 4.4 (page 8). Finally, started a FAQ section - see section 8 (page 35).

1.8.2 Added a use case that describes how to create a set of "gene models" in a manner more efficient than the current implementation within the package. As "gene model" can actually have different meanings depending on the context, I switched to call these **synthetic transcripts**; *i.e.* what we create really are synthetic transcripts combining all exons of their corresponding gene. See the use case in section 7.1 (page 25) for how this is performed. This example was first introduced at the EMBO 2013 Practical Course: Analysis of High Throughput Sequencing Data, see http://www.ebi.ac.uk/training/course/embo-practical-course-analysis-high-throughput-sequencing-data-1 for the lectures (there are links in the time-table) and here for my materials: https://umu.box.com/s/f37nl4u1p0fj4m58dxxm.

1.3.14 A new section: 6 was added describing foreseen changes to the package, see page 24. Additional changes to the vignette use-case where made thanks to Richard Friedman's suggestions, see section 7.2, page 27.

1.3.10 - 1.3.13 Some vignette discrepancies have been corrected. Thanks to Richard Friedman for spotting them, see paragraph 4.2, page 7. Some additional information about the 'gtf' and 'gff' format has been added after fixing the bug reported by Tomasz Kulinski. See section 4.1, page 6.

1.3.9 Not much change to the vignette, but for the fact that the *easyRNASeq* application note got published in **Bioinformatics** [2].

1.3.5 - 1.3.8 The vignette has been updated to report function call changes to the `easyRNASeq` function:

- the format argument defaults to bam
- the chr.sizes can be derived from the bam file header
- the addition of the knownOrganisms function
- support for variable length reads
- read files can be processed in parallel

Sections affected are: 4.1 (page 6), 4.3 (page 8), 4.3 (page 8), 4.5 (page 11), 4.10 (page 19),7.2 (page 27),7.3 (page 31)

1.3.1 - 1.3.4 The vignette has been updated to:

- first: enhance its readability.
- second: introduce a new **Use Case** section

Sections affected are: 4 (page 6), 4.1 (page 6), 4.4 (page 9), 4.6 (page 12), 5.1 (page 20), 7 (page 25)

3 Introduction

The present document describes the use of the *easyRNASeq* package to ease the processing of RNA-seq data in *R/Bioconductor*. RNA-seq [3] was introduced as a new method to perform Gene Expression Analysis, using the advantages of the high throughput of *Next-Generation Sequencing* (NGS) machines.

The goal of this vignette is, first, to show an example of the `easyRNASeq` function that generates a count table for the selected genic features of interest, *i.e.* exons, transcripts, gene models, *etc.* using the read data and the genic and genomic annotations. This overall process is described in figure 1, page 5. Shortly, the genomic and genic annotation will be retrieved from the selected/preferred source and converted into a *RangedData* object. In parallel, the sequenced reads' information (*e.g.* chromosome, position, strand, *etc.*) will be retrieved from the alignment file and, similarly, converted to a *GRanges* object. Then, reads contained in the `reads` *GRanges* object are summarized per genic annotation contained in the `annotation` *RangedData* object, which give raise to a count table that, finally, can be corrected or normalized using additional R packages.

Second, the `easyRNASeq` function arguments are detailed and additional information given on how to correct data for visualization, using *RPKM (Reads Per Kilobase of feature per Million reads in the library)* counts. Normalization can be performed by using appropriate statistical models implemented *e.g.* in the *DESeq* or *edgeR* packages. *comment: For the reasons why I call RPKM a correction and not a normalization, see [4].*

In a third part, more advanced pre-processing is described, *i.e. de-multiplexing*. Note that this step has become a standard procedure in most sequencing facilities and hence is bound to be deprecated as a function in the *easyRNASeq* package in a few release cycles.

Finally, additional post-processing such as exporting the count table as bed and wig formatted file, to be visualized into the UCSC genome browser or a stand alone genome browser are described in the vignette of the companion data package *RnaSqTutorial*. Note that the information in this tutorial is getting quickly outdated given the pace of evolution of the NGS field in general. The basic information it contains are perfectly correct though. A newer version of that tutorial is in preparation.

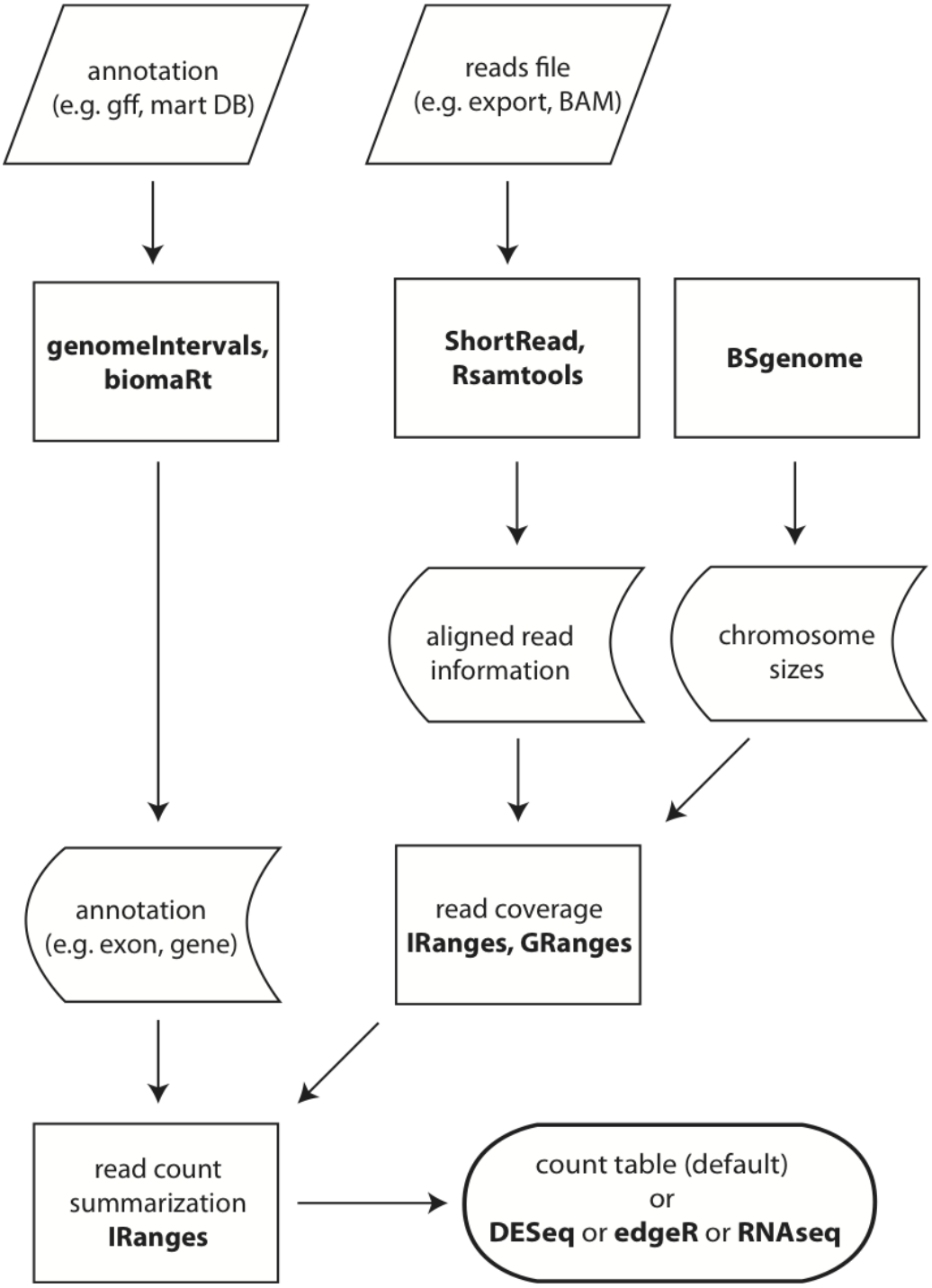


Figure 1: easyRNAseq Overview. The R packages used for the different steps are emphasized in bold face.

4 easyRNASeq

Throughout this vignette, the data present in the *RnaSeqTutorial* data package will be used to demonstrate functionalities. Refer to its documentation for more details; but shortly, it contains different *Drosophila melanogaster* RNA-Seq dataset saved in different format as well as the related annotation retrieved from FlyBase [5].

The main function of the *easyRNASeq* package is `easyRNASeq`. That should be the only processing method you need to know about when using the package. It is essentially a wrapper around other functions performing the different tasks described in Figure 1. These functions are also exported, if you feel you need to have a look at them. The *easyRNASeq* package defines an *RNAseq* class for holding the necessary information used during the processing. The `easyRNASeq` function instantiates an object of that class and although most arguments to this function are optional, it is advised to set them - especially the *readLength* and the *organismName* - to document your analysis. The `organismName` argument is actually mandatory if you want to retrieve genomic annotation using *biomaRt*; in which case, you need to provide the name as specified in the corresponding *BSgenome* package, *i.e.* "Dmelanogaster" for the *BSgenome.Dmelanogaster.UCSC.dm3* package.

warning: As there are numerous RNA-Seq protocol, each with its own specificities, the `easyRNASeq` function has a large number of arguments to cope with these. This number of arguments, with only a few defaults values might be intimidating but we are in the process of consolidating them. Meanwhile, please refer to the documentation and the examples in this vignette and the RnaSqTutorial one to get familiar with them. Do not hesitate to ask on the Bioconductor mailing list if the purpose of some argument were still unclear.

4.1 usage

Initial example In the following, the `easyRNASeq` function is called with an almost minimal set of parameters - a few optional ones too... The sequencing reads' excerpts used here were obtained from 4 fruit-fly samples subjected to a 36 cycles sequencing on an Illumina GAIIx machine. The annotation were retrieved from FlyBase and converted into an "Rdata" object.
Here, we're interested in retrieving the count table of these 4 samples for every exon in the genome.

```
> library(easyRNASeq)
> library(RnaSeqTutorial)
> count.table <- easyRNASeq(filesDirectory=system.file(
+                             "extdata",
+                             package="RnaSeqTutorial"),
+                           pattern="[A,C,T,G]{6}\\.bam$",
+                           readLength=30L,
+                           organism="Dmelanogaster",
+                           annotationMethod="rda",
+                           annotationFile=system.file(
+                             "data",
+                             "gAnnot.rda",
+                             package="RnaSeqTutorial"),
+                           count="exons"
+                           )
```

```
Checking arguments...
Fetching annotations...
```

```
Summarizing counts...
Processing ACACTG.bam
Processing ACTAGC.bam
Processing ATGGCT.bam
Processing TTGCGA.bam
Preparing output
```

```
> head(count.table)
```

```
           ACACTG.bam ACTAGC.bam ATGGCT.bam TTGCGA.bam
CG11023:1           0          0          0          0
CG11023:2           0          0          0          0
CG11023:3           0          0          0          1
CG2671:1            0          0          0          0
CG2671:2            1          0          0          1
CG2671:3           13          8         11         12
```

```
> dim(count.table)
```

```
[1] 69306     4
```

In this usage of the `easyRNASeq` function, we're reading bam files (the default `format` argument), present in the directory passed as first argument to the function (the `filesDirectory` argument) using a regular expression to match the file names (`pattern` argument). The `system.file` function retrieves the actual path on your file system where the *RnaSeqTutorial* package was installed and in that particular case of its "extdata" sub-directory. The pattern used to retrieve the files is similar to that of the *base* functions `dir` and `grep`. Here we look up files, which name contains 6 times one of the "A", "C", "G" or "T" letter, followed by the ".bam" extension. *comment: Note the use of the $ sign at the end of the regular expression to signify that the filename to be matched has to end after '.bam'. This way we prevent fetching the BAI (the BAM index) files.* Additionally, the number of sequencing cycles (`readLength`) and the `organism` that sample originates from, both optional here are detailed. Then the genomic and genic information is provided: the exons' annotation (`annotationFile` argument), which were already processed into an "RData" file (the `annotationMethod` argument). Finally, we precise that we want the sequenced reads to be summarized by "exons"; *i.e.* we want to get a read count value per exon and per sample.
And that is all. In that one command, you got the exon count table for your 4 samples!

4.2 warnings

warning: As you could see when running the previous example, warnings were emitted and quite rightly so.

1. *about the annotation: The annotation used here is redundant at two levels. First, some exons overlap; these are alternative exons from splice variants. Second, the annotation contains every splice variants, hence some exons are duplicated. Therefore counting by exons or transcripts using this annotation results in counting several reads many times and for that reason a warning is emitted, as multiple countingmay be a very significant source of error. The ideal solution is to provide an annotation object that contains no overlapping features, see the use case in section 7.1 (page 25) for an approach to reduce/control these annotation related issues. Consult as well the section 4.4 (page 8) for related counting issues.*
2. *about potential naming issue in the input file: It is not infrequent that sequencing facilities use variable naming conventions for chromosomes in comparison to those generally globally accepted for the corresponding organism. Therefore, is it not infrequent that the globally available annotation uses different chromosome names than those reported in the alignment files.*

These warnings are there to inform you about these potential issues, as they affect the accuracy of associating reads with genic or genomic features. As often misconceived, read count is not an easy task; see section 4.4 (page 8) and [6] for more details.

4.3 arguments

Going back to the easyRNASeq function, more arguments can be set to fine tune it, depending on the input at hand, the desired output, etc. Most will be detailed in the following, but not all, so check out ?easyRNASeq for more. One such argument is: filenames, that offers the possibility to give actual filenames rather than using a pattern matching approach to find the alignment files. For example, if you're only interested in two samples out of the four previously used ones, you could do:

```
> count.table <- easyRNASeq(system.file(
+                                     "extdata",
+                                     package="RnaSeqTutorial"),
+                           organism="Dmelanogaster",
+                           readLength=30L,
+                           annotationMethod="rda",
+                           annotationFile=system.file(
+                             "data",
+                             "gAnnot.rda",
+                             package="RnaSeqTutorial"),
+                           count="exons",
+                           filenames=c("ACACTG.bam", "ACTAGC.bam"))
```

Several arguments influence the easyRNASeq function. These can be grouped in 5 categories concerning:

- the input
- the annotations
- the counts summarization
- the output
- the optional correction/normalization

Input Using the format argument, reads can be loaded from BAM files (default) or any format supported by the *ShortRead* [7] package. The SAM/BAM format [8] is becoming a *de-facto* standard for storing NGS aligned data. Gapped alignment can be read from BAM files, set the gapped argument to "TRUE" for that. *comment: Note that as BAM is now the standard for storing read alignments, easyRNASeq support of other formats - through the use of ShortRead will be abandoned in a few release cycles.*

4.4 Critical information regarding the input

warning: As previously mentioned, associating reads with features is not straightforward, and numerous factors have to be taken into account that affect the counting accuracy. First, it is critical to know what alignments where reported by the aligner used. Does the alignment file contains multi-mapping reads; i.e. are reads aligning to numerous loci reported? Second, it is also critical to understand the gene's annotation in detail. Is it from a well-established model organism? Are there many splicing isoforms reported per gene? How do these differ; do they have different exons, different UTRs, etc. ? Understanding pitfalls and caveats from both will help you decide what settings are required within the frame of your experiment. In most common RNA-Seq experiment, the goal is to look for differential expression between 2 conditions within one organism. The most stringent approach to that is:

1. *to use only uniquely mapping reads and discarding others - this is the approach taken by e.g. HTSeq; see http://www-huber.embl.de/users/anders/HTSeq/doc/overview.html.*
2. *to ensure that reads can be assigned to a single feature only - i.e. when a read cannot be confidently assign to a feature; e.g. if a read locus overlap two genes either neighbouring or on opposite strand - discard them. This can be achieved by 2 means, either by assigning reads to features first and then filtering the results or filtering the annotation prior to assigning the reads to prevent any such feature overlap.*

As of now, it is important to know that easyRNASeq just reports warnings if these conditions are unmet. Obtaining the correct set of alignments for your analysis is specific to your aligner of choice and hence cannot be described here. Obtaining a set of synthetic transcripts that would fully remove or diminish the occurence of ambiguous read assignment is detailed in the first use case of this vignette, see section 7.1 (page 25).

Annotation The `easyRNASeq` function currently accepts the following `annotationMethods`:

- "biomaRt" use the *biomaRt* [9] package to retrieve the annotation
- "env" use a *RangedData* or *GRangesList* class object present in the environment - see page 30 for a use case.
- "gff" reads in a gff version 3 file
- "gtf" reads in a gtf file as provided by Ensembl [10]
- "rda" load an RData object. The object needs to be named `gAnnot` and of class *RangedData* or *GRangesList*. See the use case page 29 to prepare such an object using the *GenomicFeatures*. ".rda" is the filename extension of serialized R data object (*i.e.* object written to disk using the `save` function). The filename extension for these files used to be and still is sometimes".RData". The actual filename extension used is of no importance to `easyRNASeq` - see page 29 for a use case.

The structure of the *RangedData* object that has to be provided when using the "rda" or "env" `annotationMethods` is described in the section 4.6 (page 12). For using a *GRangesList*, see the use case page 29. More details on the gff3 and gtf format can be obtained from:

- gff3: http://www.sequenceontology.org/gff3.shtml
- gtf: http://mblab.wustl.edu/GTF22.html

Unlike for the gtf, the gff3 ninth column holding the attributes has no mandatory tags. Some of them are however "predefined". As exemplified further down (see 4.6, 12), we expect these "predefined" attributes to be used and we rely on the "ID", "Parent" and "Name" to identify the relation between a gene and its exons and transcripts.

Summarization The reads can be counted/summarized by:

- exons
- features (any features such as introns, enhancers, *etc.*)
- transcripts
- geneModels: they consists of the set of disjoint "synthetic" exons that fully overlap every known exons and UTRs of a gene; *I.e.* every alternate exon will be split into separate "synthetic" exons and these exons will be grouped into a set that correspond to a single gene.

When you use the "geneModels" summarization, no reads will be counted twice at the exception of those mapping overlapping exons of different genes, whether on the same strand or not. The `easyRNASeq` function does not deal with these situation as it does not yet support stranded reads, hence as previously a warning will be emitted if this occurs. The implementation of the geneModel generation is actually outdated - it is rather slow - and is being re-implemented as described in section 7.1 (page 25). Note that the geneModel summarization as it is now, is deprecated.

Output The results can be exported in five different formats:

- count table (the default, a n (features) x m (samples) `matrix`).
- a *DESeq* [11] *CountDataSet* class object. Useful to perform further analyses using the *DESeq* package.
- an *edgeR* [12] *DGEList* class object. Useful to perform further analyses using the *edgeR* package.
- an *RNAseq* class object. Useful for performing additional pre-processing without re-loading the reads and annotations.
- as an instance of the *RangedSummarizedExperiment* class. See section 6 (page 24) for additional details.

Correction The results can optionally be "corrected" as *Reads per Kilobase of feature per Million reads in the library* (RPKM, [3]), useful for visualization but not for differential expression, see []. *comment: Note that the `vst` and `voom` approaches from the DESeq2 and limma packages, respectively offer more powerful approaches for correcting the data prior to visualization. These will be included in future version of the easyRNASeq package.*

Normalization The normalization step is also available when the results are exported in the *DESeq* or *edgeR* formats. This generates plots to assess the validity of the normalization and help decide how to proceed with any downstream analysis.

4.5 different input

The default input format of the easyRNASeq function is the BAM format, as exemplified before. It does at the moment support older formats through the *ShortRead* package, but as BAM is nowadays standard, this support will eventually be phased out in a few release cycles.

In the first example, we read a BAM file containing gapped alignments. In that particular case, TopHat [13] was used to look for splicing events. Note the use of the gapped boolean argument.

```
> count.table <- easyRNASeq(system.file(
+                                       "extdata",
+                                       package="RnaSeqTutorial"),
+                           organism="Dmelanogaster",
+                           readLength=36L,
+                           annotationMethod="rda",
+                           annotationFile=system.file(
+                             "data",
+                             "gAnnot.rda",
+                             package="RnaSeqTutorial"),
+                           gapped=TRUE,
+                           count="exons",
+                           filenames="gapped.bam")
```

The next example show how to load an Illumina 'export' file, which was Illumina's default output before the CASAVA v1.8 release in spring 2011. The `format` argument is set to "aln" and an additional argument: `type` is provided that precise the input file format. See the *ShortRead* `readAligned` help page for all available formats. A final argument: the `filter` argument, essential to filter usable reads is also given. This argument is crucial as export files contain every read, including those not aligning and those not passing the original quality filter performed during the Illumina Base Calling, a.k.a. the chastity filter. For more details, see the section "Filtering the Data" in the *RnaSqTutorial* vignette. Finally, also note the `chr.sizes` argument; it precises the sizes of the chromosomes using a named integer vector.Indeed, these cannot be determined programatically unlike for the BAM format where the chromosome sizes can be extracted from the header.

```
> count.table <- easyRNASeq(system.file(
+                                       "extdata",
+                                       package="RnaSeqTutorial"),
+                           organism="Dmelanogaster",
+                           chr.sizes=seqlengths(Dmelanogaster),
+                           readLength=36L,
+                           annotationMethod="rda",
+                           annotationFile=system.file(
+                             "data",
+                             "gAnnot.rda",
+                             package="RnaSeqTutorial"),
+                           format="aln",
+                           type="SolexaExport",
+                           count="exons",
+                           pattern="subset_export",
+                           filter=compose(
+                             chromosomeFilter(regex="chr"),
+                             chastityFilter()))
```

4.6 different annotation

This sections describes how to use different annotation sources to retrieve genic information. It also describes the expected format of these annotations.

comment: Important Note: there are many derivative of the GFF3 and GTF file formats, some of which not strictly applying the convention these files should implement. Using incorrectly formatted file is bound to fail, but most of the time a relevant error message will be reported. GTF files as produced by e.g. Ensembl and GFF3 files as produced by e.g. Flybase are correct implementation of these formats. If you encounter problems with an annotation file, please try first to locate and correct them. For the GFF format, the example file present in the RnaSqTutorial package can be used as a reference. If you have installed this package the following gives you that gff file path on your machine:

```
> system.file(
+             "extdata",
+             "Dmel-mRNA-exon-r5.52.gff3",
+             package="RnaSeqTutorial")
```

comment: If you do not manage to identify the issue, please contact the Bioconductor mailing list for help.

biomaRt The following is an example of how to use biomaRt to retrieve annotation.

```
> rnaSeq <- easyRNASeq(system.file(
+                                  "extdata",
+                                  package="RnaSeqTutorial"),
+                          organism="Dmelanogaster",
+                          readLength=36L,
+                          annotationMethod="biomaRt",
+                          gapped=TRUE,
+                          count="exons",
+                          filenames="gapped.bam",
+                          outputFormat="RNAseq")
```

Once the annotation is fetched, it can be retrieved provided that the `outputFormat` argument was set to *RNAseq*. This implies a complete S4 object of class *RNAseq* is returned and not only the count table (see 4.7 (page 15) for details on the different output). The `genomicAnnotation` accessor allows then to access the annotation stored in the *RNAseq* object.

```
> gAnnot <- genomicAnnotation(rnaSeq)
```

The `gAnnot` - of class *RangedData* - can then be saved in a '.rda' file for a faster re-processing using the `easyRNASeq` function with the arguments `annotationMethod` and `annotationFile` set to "rda" and to the 'rda' filepath, respectively. *comment: Note, however that it is necessary to save that object under the name "gAnnot".* Finally, it would be important to pre-process that new annotation in the manner described in section 7.1 (page 25) for reason described in section 4.4 (page 8)

genomeIntervals The next example shows how to perform the same provided that you have a gtf or gff3 formatted file at hand. While the gtf format is more constrained, gff formatted file will likely have varying gff attributes (its ninth last column). This column contains "key=value" pairs separated by semi-colons. We depend on specific keys to be present and these are *ID*, *Parent*, and *Name*, all strongly suggested by the gff3 format specification. When parsing a gff3 file, we're only considering feature of type 'exon' (the gff3 third column). Then, we expect the *ID* key to have the format: *geneID:exonNumber*; both the exon and gene annotation being derived from it. The *Parent* key should contain the transcript ID. If one exon is part of several transcripts, these can be concatenated using commas without space. Finally, the *Name*

key should contain the gene name, but its presence is actually facultative for the processing. As we are using the `readGff3` function from the *genomeIntervals* package, it is as well essential that the key=value pairs are separated by semi-colons **whitout** space. Have a look at the 'annot.gff' file used in the next code chunck for an example of a gff3 file formatted as the `easyRNASeq` expects it.

```
> count.table <- easyRNASeq(system.file(
+                                       "extdata",
+                                       package="RnaSeqTutorial"),
+                           organism="Dmelanogaster",
+                           readLength=36L,
+                           annotationMethod="gff",
+                           annotationFile=system.file(
+                             "extdata",
+                             "Dmel-mRNA-exon-r5.52.gff3",
+                             package="RnaSeqTutorial"),
+                           gapped=TRUE,
+                           count="exons",
+                           filenames="gapped.bam")
```

RangedData Internally, the `easyRNAseq` function converts the retrieved annotation - when executed with the `annotationMethod` argument "gff", "gtf", or "biomaRt" - into an object of the *RangedData* class. Using the `annotationMethod` argument "env" or "rda" requires that you provide such an object. The following example shows the expected structure of that object.

```
> library(IRanges)
> gAnnot <- RangedData(
+                      IRanges(
+                              start=c(10,30,100),
+                              end=c(21,53,123)),
+                           space=c("chr01","chr01","chr02"),
+                           strand=c("+","+","-"),
+                           transcript=c("trA1","trA2","trB"),
+                           gene=c("gA","gA","gB"),
+                           exon=c("e1","e2","e3"),
+                           universe = "Hs19"
+                           )
> gAnnot
```

```
RangedData with 3 rows and 4 value columns across 2 spaces
     space     ranges |      strand  transcript        gene        exon
  <factor>  <IRanges> | <character> <character> <character> <character>
1    chr01 [ 10,  21] |           +        trA1          gA          e1
2    chr01 [ 30,  53] |           +        trA2          gA          e2
3    chr02 [100, 123] |           -         trB          gB          e3
```

In that object, we describe the genomic location of three exons (chromosome, position and strand) as well as their transcript and gene membership. Nothing more is required. Note that the names' spelling is **essential** for the `easyRNAseq` function to work properly. Indeed, the `count` argument will be used to lookup the proper values in the annotation, *e.g.* for summarizing by "exons", an "exon" column in the *RangedData* object is mandatory; a "feature" one for counting "features", a "transcript" one for "transcripts", *etc.* . For a peek at the *RangedData* class `gAnnot` object used in the *RnaSeqTutorial* tutorial, do:

```
> library(RnaSeqTutorial)
```

```
> data(gAnnot)
> gAnnot
```

GRangesList The *easyRNASeq* package support as well annotation provided as a *GRangesList* class object (from the *GenomicRanges* package). Converting a *RangedData* class object into a *GRangesList* class object is pretty straightforward.

```
> grngs <- as(gAnnot,"GRanges")
> grngsList<-split(grngs,seqnames(grngs))
> grngsList
```

```
GRangesList object of length 2:
$chr01
GRanges object with 2 ranges and 3 metadata columns:
      seqnames    ranges strand |  transcript        gene        exon
         <Rle> <IRanges>  <Rle> | <character> <character> <character>
  [1]    chr01  [10, 21]      + |        trA1          gA          e1
  [2]    chr01  [30, 53]      + |        trA2          gA          e2

$chr02
GRanges object with 1 range and 3 metadata columns:
      seqnames     ranges strand | transcript gene exon
  [1]    chr02 [100, 123]      - |        trB   gB   e3

-------
seqinfo: 2 sequences from Hs19 genome; no seqlengths
```

The advantage of doing so is that the *RangedData* class might get deprecated in the future. The next advantage is that the *GRangesList* class is strand-aware.

Remember *comment: Generating the proper annotation is probably the most important step in processing your RNA-Seq sample. Mind the warnings produced by the easyRNASeq function, they might be annoying, but there are there for a good reason: to help.*

4.7 different output

matrix This is the default output - a matrix of m features by n samples - that has been exemplified throughout the beginning of this vignette.

RNAseq This has also been introduced previously, see section 4.6 (page 12), but shortly - as shown below - the only change that need doing is to use the *RNAseq* value of the `outputFormat` argument.

```
> rnaSeq <- easyRNASeq(system.file(
+                                  "extdata",
+                                  package="RnaSeqTutorial"),
+                      organism="Dmelanogaster",
+                      readLength=30L,
+                      annotationMethod="rda",
+                      annotationFile=system.file(
+                        "data",
+                        "gAnnot.rda",
+                        package="RnaSeqTutorial"),
+                      count="exons",
+                      pattern="[A,C,T,G]{6}\\.bam$",
+                      outputFormat="RNAseq")
```

```
Checking arguments...
Fetching annotations...
Summarizing counts...
Processing ACACTG.bam
Processing ACTAGC.bam
Processing ATGGCT.bam
Processing TTGCGA.bam
Preparing output
```

CountDataSet, DGEList, RangedSummarizedExperiment More details on how to generate *CountDataSet* (*DESeq*) or *DGEList* (*edgeR*) are given in section 4.9 (page 17).
The integration of the *SummarizedExperiment* `outputFormat` in the *easyRNASeq* package is recent, but at term it will be the new `outputFormat` default. See section 6 (page 24) for details.

4.8 different summarization

As introduced previously, there are four possible counting method accesible through an `easyRNASeq` call: by "exons", "features", "transcripts" and "genes". However, to perform multiple summarization on the same data, *e.g.* by "exons" and "transcripts", calling the `easyRNASeq` twice is inefficient as the read files and the annotation twice would have to be processed twice. To avoid this, specific functions are available, which are applicable to an object of class *RNAseq*, see details about that class in section 4.7 (page 15). These functions are:

1. `exonCounts`
2. `featureCounts`
3. `transcriptCounts`
4. `geneCounts` This function takes an additional parameter defining the kind of gene summarization, either `bestExons` or `geneModels`. The `bestExons` summarization will return per gene, the highest exon count. The `geneModels` summarization first calculate a gene model and then return the read count for it. Rather than using the genes / geneModels paradigm, consider using synthetic transcripts as described in section 7.1 (page 25).
5. `readCounts` a function to access the different count tables stored in the *RNAseq* object.

For example, to summarize the reads per transcripts, apply the `transcriptCounts` function on the previously generated `rnaSeq` object - see section 4.7 (page 15) - and use the `readCounts` function to access the transcript count table.

```
> rnaSeq <- transcriptCounts(rnaSeq)
> head(readCounts(rnaSeq,'transcripts'))
```

```
            ACACTG.bam ACTAGC.bam ATGGCT.bam TTGCGA.bam
FBtr0005009          0          0          0          0
FBtr0005088         56         29         44         52
FBtr0005673          2          0          1          2
FBtr0005674          7          3          9          6
FBtr0006151          0          0          0          1
FBtr0070000          0          0          0          0
```

Summarizing by transcript is frequently inherently biased as exons are often part of different splicing variants. For that reason, it might be better to summarize the data per genes - or synthetic transcripts- as described in section 7.1 (page 25).

4.9 optional correction or normalization

In this section, different count *correction* and *normalization* are described. First the RPKM, a common sense correction that take the genic feature size (exon, transcript, gene model,...) and the total number of reads sequenced in the library into account. Its use is not recommended for doing any kind of differential expression analysis - see [4] - but is adequate for visualization; *e.g.* to create tracks to be displayed in a genome browser. Actually for state of the art visualization, one can use a `voom` (from the *limma* package) or `vst` (from the *DESeq2* package) approach.

Here is an example on how to get a RPKM count table:

```
> count.table <- easyRNASeq(system.file(
+                                       "extdata",
+                                       package="RnaSeqTutorial"),
+                           organism="Dmelanogaster",
+                           readLength=30L,
+                           annotationMethod="rda",
+                           annotationFile=system.file(
+                             "data",
+                             "gAnnot.rda",
+                             package="RnaSeqTutorial"),
+                           count="exons",
+                           filenames=c("ACACTG.bam", "ACTAGC.bam",
+                             "ATGGCT.bam", "TTGCGA.bam"),
+                           normalize=TRUE
+                           )
```

In addition, easyRNASeq generated raw count tables can be post-corrected using the `RPKM` method:

```
> feature.size = width(genomicAnnotation(rnaSeq))
> names(feature.size) = genomicAnnotation(rnaSeq)$exon
> lib.size=c(
+   "ACACTG.bam"=56643,
+   "ACTAGC.bam"=42698,
+   "ATGGCT.bam"=55414,
+   "TTGCGA.bam"=60740)
> head(RPKM(readCounts(rnaSeq,summarization="exons")$exons,
+           NULL,
+           lib.size=lib.size,
+           feature.size=feature.size))
```

```
           ACACTG.bam ACTAGC.bam ATGGCT.bam TTGCGA.bam
CG11023:1          0          0          0          0
CG11023:2          0          0          0          0
CG11023:3          0          0          0         20
CG2671:1           0          0          0          0
CG2671:2         130          0          0        121
CG2671:3         295        241        255        254
```

The same can be directly done on *RNAseq* class objects.

```
> head(RPKM(rnaSeq,from="transcripts"))
```

```
             ACACTG.bam ACTAGC.bam ATGGCT.bam TTGCGA.bam
FBtr0005009        0.0          0        0.0        0.0
```

```
FBtr0005088      338.4        233      272.1      292.9
FBtr0005673        7.5          0        3.9        7.0
FBtr0005674       21.9         12       28.8       17.5
FBtr0006151        0.0          0        0.0        6.6
FBtr0070000        0.0          0        0.0        0.0
```

For normalizing the data, numerous solution are available in *Bioconductor*, see [4] for details. Only the *edgeR* and *DESeq* packages are currently imported by *easyRNASeq*.

DESeq To be able to normalize the data using *DESeq* (or *edgeR* for that matter), one needs to define the samples' "conditions", *e.g.* "disease" vs. "healthy". To ensure traceability, the *easyRNASeq* package require the conditions to be a *named vector* where the names are the raw data filenames.

```
> conditions=c("A","A","B","B")
> names(conditions) <- c("ACACTG.bam", "ACTAGC.bam",
+                        "ATGGCT.bam", "TTGCGA.bam")
```

Once the `conditions` has been defined, pass it to the `conditions` argument of the `easyRNASeq` function and set the `outputFormat` argument to DESeq. An additional `normalize` argument is available to trigger or not the count normalization implemented in the *DESeq* package. In either case, a `CountDataSet` object is returned. Additional argument to the *DESeq* function can be provided through the ... arguments of the `easyRNASeq` function. *e.g.* in the following example, we precise the kind of fit (*local*) that needs to be performed by the `estimateDispersion` function of the *DESeq* package. Since we have few data, the default fit fails and reports an error telling us to change the `fitType` argument.

```
> countDataSet <- easyRNASeq(system.file(
+                                        "extdata",
+                                        package="RnaSeqTutorial"),
+                            organism="Dmelanogaster",
+                            readLength=30L,
+                            annotationMethod="rda",
+                            annotationFile=system.file(
+                              "data",
+                              "gAnnot.rda",
+                              package="RnaSeqTutorial"),
+                            count="exons",
+                            filenames=c("ACACTG.bam", "ACTAGC.bam",
+                              "ATGGCT.bam", "TTGCGA.bam"),
+                            normalize=TRUE,
+                            outputFormat="DESeq",
+                            conditions=conditions,
+                            fitType="local"
+                            )
```

comment: Note that setting the `normalize` argument to `TRUE` generates diagnostics plots as detailed in the DESeq vignette. The plot produced in the present examples are irrelevant as the dataset is too small. You can turn the plotting off by setting the `plot` argument to `FALSE`. Have a look at the DESeq vignette (essential if you plan to use DESeq anyway!) for the plot explanation.

edgeR The same procedure can be done using the *edgeR* package functionalities.

```
> dgeList <- easyRNASeq(system.file(
+                                   "extdata",
```

```
+                                   package="RnaSeqTutorial"),
+                         organism="Dmelanogaster",
+                         readLength=30L,
+                         annotationMethod="rda",
+                         annotationFile=system.file(
+                           "data",
+                           "gAnnot.rda",
+                           package="RnaSeqTutorial"),
+                         count="exons",
+                         filenames=c("ACACTG.bam", "ACTAGC.bam",
+                           "ATGGCT.bam", "TTGCGA.bam"),
+                         normalize=TRUE,
+                         outputFormat="edgeR",
+                         conditions=conditions
+                         )
```

As for the former paragraph about *DESeq* (see section 4.9 (page 18)), the diagnostics plots generated are only semi-relevant. Check out the *edgeR* manual for more details about these plots. Note that producing the plots is rather slow.

Next steps At this stage you are done with the normalization and what's ahead of you: calling differential expression, exporting track files for visualization, etc. is not the scope of the *easyRNASeq* package. This one has a few more functionalities, the most important of which will be described in the next section. To proceed with your data analysis, check the relevant package vignettes (*DESeq*, *edgeR*) for differential expression analysis and the *RnaSqTutorial* for examples of track files generation using the *rtracklayer* package.

4.10 Performance

The geneModel generation, the read counting and summarization can be parallelized. Use the `nbCore` argument to set the number of CPU cores to use. There are a number of word of caution:

1. CPU cores means CPU cores, *i.e.* located on the same physical machine. Do not expect it to work across machines.
2. there's no load nor memory management, so watch out yourself for it. Memory will scale linearly with the number and size of read libraries (e.g. bam files) you process in parallel.
3. It's using the R 'parallel' package, so it's supported on all three main OS. However, some cosmetic reporting might get lost.

5 Advanced usage

5.1 De-multiplexing samples

Nowadays, NGS machines produces huge quantity of"raw" reads (40M to 150M reads per Illumina MiSeq or Illumina HiSeq lanes respectively), that the coverage obtained per lane for the transcriptome of "small" genome-sized organisms is for a single sample essentially a waste of resource. *e.g.* one lane of HiSeq results in an approximate 3,000X coverage of the *Saccharomyces cerevisiae* genome which is 12Mb large. Therefore, techniques to have several samples running as a *single* library have been created [14, 15], using 4-6bp barcodes to uniquely identify samples; this is called **multiplexing**. A 30X coverage per sample of 48 yeast samples can hence be obtained froman average Illumina GenomeAnalyzer GAIIx run (105bp read, single end). This approach is very advantageous to researchers, especially in term of costs, but it adds an additional layer of pre-processing that is not as trivial as one may think. Extracting the barcodes would be fairly straightforward, but for the average 0.1 percent sequencing error rate that introduces a lot of multiplicity in the actual barcodes present in the samples. A proper design of the barcodes, maximizing the Hamming distance[16, 17] is an essential step for a proper de-multiplexing.
There are two kinds of barcoding, the one described in [14] where the barcode is part of the read sequence and the one developed by Illumina, where the barcode is read in a separate sequencing reaction after the first mate sequencing.
The data used in the following example has been sequenced using the Illumina protocol. We'll look at the specificities that this introduce.

comment: This pre-processing procedure has to be applied on the raw reads before any alignment is performed. Most often, one would use the "fastq" formatted file as input. In the particular case of the Illumina protocol, the barcodes can be retrieved from the fastq ID lines or from the Illumina export format. The export format is normally the result of aligning the reads with ELAND, the Illumina aligner, but it can be generated as well without aligning the reads, *i.e.* the export file may not contain any alignment information. *comment: The ShortRead package offers a quick functionality to access the barcode field of an export file, which we take advantage of in the next example; the `withAll` argument of the `readAligned` function.*

```
> alns <- readAligned(
+                     system.file(
+                                 "extdata",
+                                 package="RnaSeqTutorial"),
+                     pattern="multiplex_export",
+                     filter=compose(
+                       chastityFilter(),
+                       nFilter(2),
+                       chromosomeFilter(regex="chr")),
+                     type="SolexaExport",
+                     withAll=TRUE)
```

Note the use of the `withAll` argument. It is essential to get the barcode, since this data was multiplexed using the Illumina protocol. For the Illumina protocol, the barcode is read in a separate sequencing reaction and its sequence is reported as a field of the *export* file. This field is not parsed by default by the `readAligned` function to save time and memory. It becomes available when the data is loaded using the `withAll` argument and is afterwards accessible in the metadata of the returned object. It can be accessed using the following command:

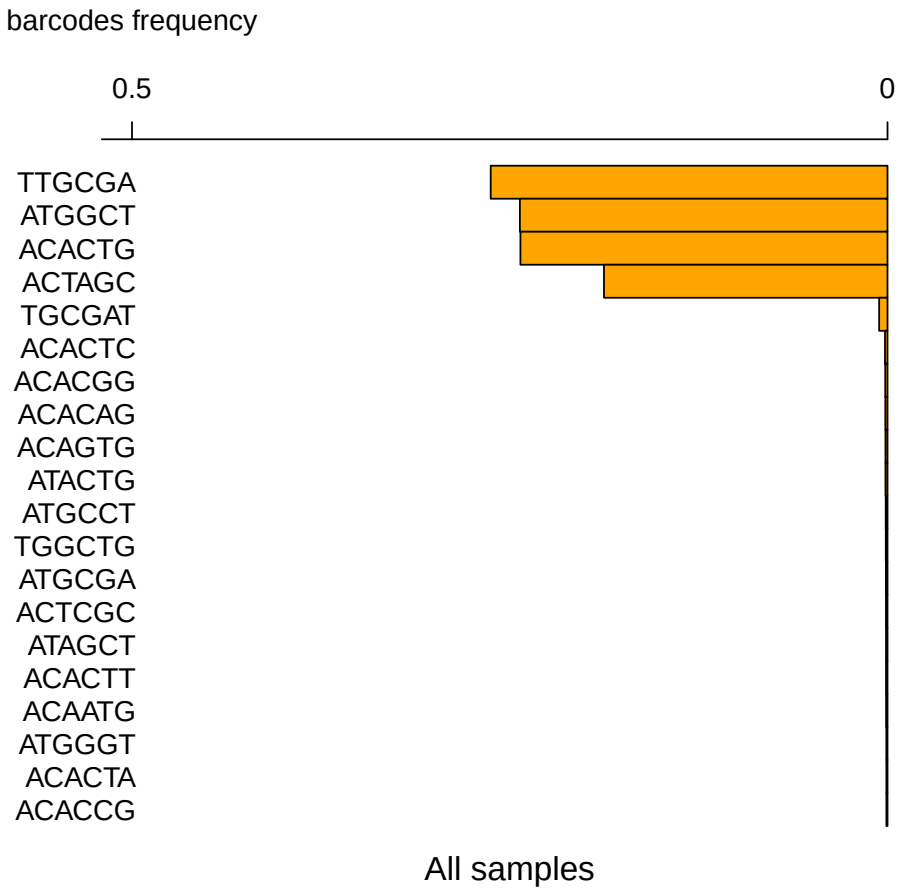
```
> alignData(alns)$multiplexIndex
```

where `alns` is the object of the *ShortRead* *AlignedRead* class.

In the following, we will look at the other kind of barcoding, where the barcode is part of the sequenced sequence. First, we will create some plots to evaluate the barcoding efficiency.

```
> barcodes=c("ACACTG","ACTAGC","ATGGCT","TTGCGA")
```

```
> barcodePlot(alns,
+             barcodes=barcodes,
+             type="within",
+             barcode.length=6,
+             show.barcode=20,
+             main="All samples",
+             xlim=c(0,0.5))
```



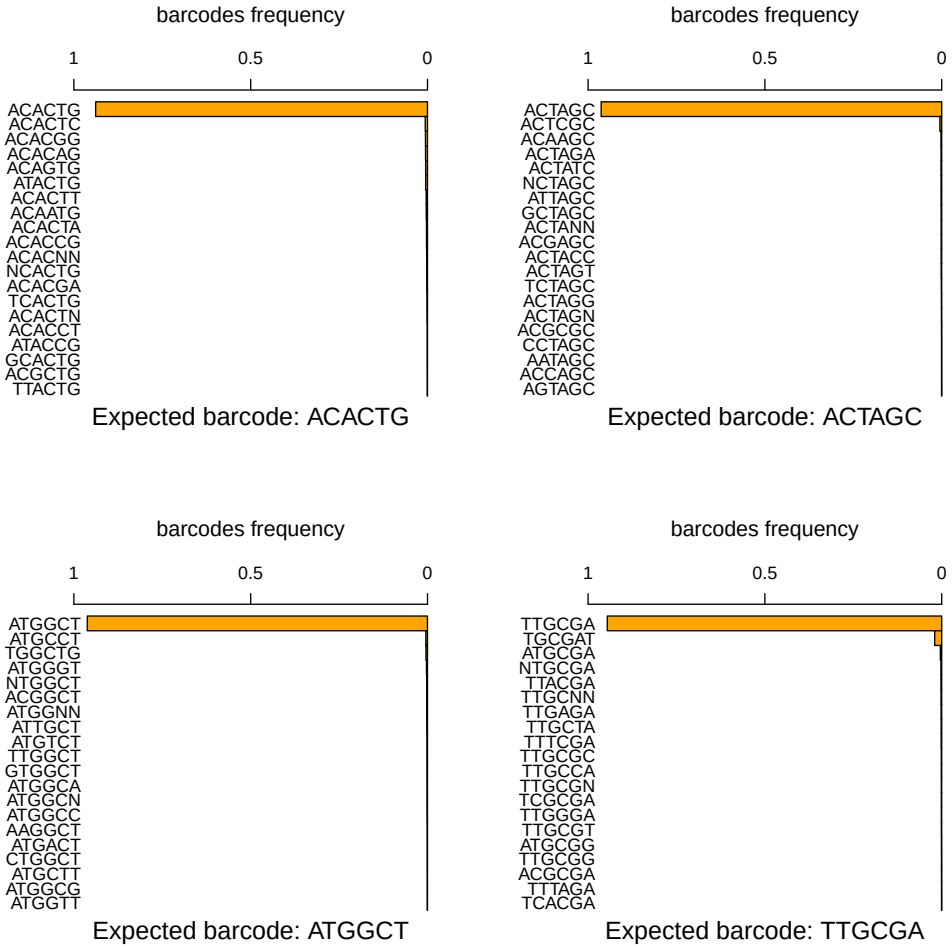
All the barcodes seem to be almost equally distributed. Every one has a proportion close to 25%. Just the "ACTAGG" seems to have been either less amplified during the library preparation or simply had a lower concentration than the other or generated less clusters. Overall, only a low percentage of them cannot be surely assigned. Once this has been verified, the sample can be *demultiplexed*.

```
> dem.alns <- demultiplex(alns,
+                         barcodes=barcodes,
+                         edition.dist=2,
+                         barcodes.qty=4,
+                         type="within")
```

```
> dem.alns$reads[[1]]
> dem.alns$barcodes[[1]]
```

There remain to write the extracted data to file, or proceed it within R. Again performing some validation plot is important to ensure that the *de-multiplexing* process succeeded.

```
> par(mfrow=c(2,2))
> barcode.frequencies <- lapply(
+                               names(dem.alns$barcodes),
+                               function(barcode,alns){
+                                 barcodePlot(
+                                             alns$barcodes[[barcode]],
+                                             barcodes=barcode,
+                                             type="within",barcode.length=6,
+                                             show.barcode=20,
+                                             main=paste(
+                                               "Expected barcode:",
+                                               barcode))
+                               },dem.alns)
```



In the previous figure, we can observe the demultiplexed barcodes. It is important to assess wether within a given sample there was a barcode bias. Generally, visually assessing the data is important to make sure that the raw data you are looking at agrees with the experimental design that was performed. This is discussed in the `RnaSeqTutorial` vignette of the *RnaSqTutorial*.

Finally, we can save the demultiplexed files to disk.

```
> status <- lapply(
+                   names(dem.alns$barcodes),
+                   function(barcode,alns){
+                     writeFastq(
+                                 alns$reads[[barcode]],
+                                 file=paste(
+                                   barcode,
+                                   "fastq",sep="."))
+                   },dem.alns)
```

The fastq files are now ready to be aligned with your preferred aligner - alternatively, other R packages such as *Rsubread*, *gmapR*, *QuasR* can be used to align them - and the resulting bam files processed with the easyRNASeq function as described in the first part of the present vignette! Enjoy.

6 Yet to come

The easyRNASeq can now return a *RangedSummarizedExperiment* object. The aim is to have it be the only output in the next development version of the *easyRNASeq* package (version $1.9.x$). This in order to consolidate the kind of objects used for Next-Generation Sequencing in the Bioconductor repository.

```
> ## creating a RangedSummarizedExperiment from 4 bam files
> sumExp <- easyRNASeq(filesDirectory=system.file(
+                   "extdata",
+                   package="RnaSeqTutorial"),
+                 pattern="[A,C,T,G]{6}\\.bam$",
+                 readLength=30L,
+                 organism="Dmelanogaster",
+                 annotationMethod="rda",
+                 annotationFile=system.file(
+                   "data",
+                   "gAnnot.rda",
+                   package="RnaSeqTutorial"),
+                 count="exons",
+                     outputFormat="SummarizedExperiment"
+                 )
> ## the counts
> assays(sumExp)
> ## the sample info
> colData(sumExp)
> ## the 'features' info
> rowRanges(sumExp)
```

See the *RangedSummarizedExperiment* class defined in the *SummarizedExperiment* package for more details on last three accessors used in the example.

Support for *DESeq2* is also planned for version $1.10.x$ as well as additional visualization taking advantage of the voom/vst transformations.

7 Use Cases

The following use cases have been created to answer user request from the Bioconductor mailing list. We want to thank Francesco Lescai, Wade Davis and Richard Friedman for asking pertinent questions that helped us make this vignette better.
The first use case shows how to create an annotation that contains synthetic transcripts; *i.e.* a transcript combining every single exons of a gene into a single abiological splice-variant. The aim is to mesure expression at the gene level, while avoiding counting biases that may be introduced when working with multiple splice variants sharing common exons. If you are interested in such splice variants expression, you need to look for more appropriate analysis expression pipeline (RSEM, MMSEQ, BitSeq, *etc.*)
The second use case examplifies how to use the easyRNASeq to get *DESeq* normalized data from two human samples. It introduces as well how to use the *GenomicFeatures* package to retrieve annotations. Note that the data for this example is not readily available. If you want to reproduce this example, you will need to get 2 (at least) fastq files from either the SRA or ENA websites and align them against the human genome using your aligner of choice and convert the alignment into the BAM format. An example of how to achieve this in R is described in the beginning of the use case.
The third use case describe how to combine different annotation (chromosomic and genic), when for example the chromosome names in the aligned file(s) are different from the annotation retrieved using *(*biomaRt. In both use case, we will assume the *easyRNASeq* library as already been loaded as in:

```
> library(easyRNASeq)
```

7.1 Creating a set of synthetic transcripts

The gff3 is retrieved from FlyBase [5]: ftp://ftp.flybase.net/releases/FB2013_06/dmel_r5.54/gff/dmel-all-no-analysis-r5.54.gff.gz. As the file is rather large, only the line containing "gene", "mRNA" and "exon" are kept using an *awk* script.

```
awk '{if($3=="gene"){print}; if($3=="mRNA"){print}; if($3=="exon"){print}}'
                          < dmel-all-no-analysis-r5.54.gff > dmel_r5-54.gff3
```

Set your working environment to the directory where you created dmel_r5-54.gff3.

```
> ## lib - loaded by easyRNASeq already
> library(genomeIntervals)
> ## read in the gff
> gff <- readGff3("dmel_r5-54.gff3")
> ## look at the gff
> gff
> nrow(gff[gff$type=="exon",])
> nrow(gff[gff$type=="mRNA",])
> nrow(gff[gff$type=="gene",])
> ## map the mRNA ID to the gene ID
> transcriptGeneMapping <- data.frame(
+   getGffAttribute(gff[gff$type=="mRNA",],"ID"),
+   getGffAttribute(gff[gff$type=="mRNA",],"Parent"))
> head(transcriptGeneMapping)
> ## extract the exons IRangesList
> ## ie exons are grouped by gene
> ## and their coordinates are stored as an IRanges object
> sel <- gff$type=="exon"
> rngList<- split(IRanges(start=gff[sel,1],end=gff[sel,2]),
```

```
+                 transcriptGeneMapping[match(
+                   sapply(strsplit(
+                     getGffAttribute(gff[sel,],"Parent"),
+                     ","),"[",1),
+                   transcriptGeneMapping$ID),"Parent"])
> rngList
> ## check what's the gene with the max number of exons
> mostExons <- rev(names(table(elementLengths(rngList))))[1]
> mostExons
> ## work the magic; collapse the genes IRanges
> rngList<- reduce(rngList)
> rngList
> ## what's the max now?
> rev(names(table(elementLengths(rngList))))[1]
> ## create the gff
> ## get the gff information; here we simply duplicate the
> ## first exon of every gene by the number of synthetic
> ## exons per gene. The content will be updated afterwards.
> exons <- gff[sel,]
> syntheticGeneModel<- exons[rep(
+   match(names(rngList),
+   transcriptGeneMapping[
+     match(sapply(
+       strsplit(getGffAttribute(exons,"Parent"),
+                ","),"[",1),
+     transcriptGeneMapping$ID),"Parent"]),
+   elementLengths(rngList)),]
> ## update the coordinates
> syntheticGeneModel[,1]<- unlist(start(rngList))
> syntheticGeneModel[,2]<- unlist(end(rngList))
> ## change the source
> levels(syntheticGeneModel$source)<- "inhouse"
> ## get the exon number for the minus strand
> exonNumber<- lapply(elementLengths(rngList),":",1)
> ## reverse them on the plus strand
> sel<- strand(syntheticGeneModel)[cumsum(elementLengths(rngList))] == "+"
> exonNumber[sel]<- sapply(exonNumber[sel],rev)
> ## update the attributes
> syntheticGeneModel$gffAttributes<- paste("ID=",
+   rep(names(rngList),elementLengths(rngList)),
+   ":",unlist(exonNumber),";Parent=",
+   rep(names(rngList),elementLengths(rngList)),".0",sep="")
> ## write the file
> writeGff3(syntheticGeneModel,file="dmel_synthetic_transcript_r5-54.gff3")
```

Now we can use the 'gff3' file as our annotation. As we have created synthetic transcripts, we can directly use the 'transcripts' value for the *count* argument of the easyRNASeq function as follows:

```
> sumExp <- easyRNASeq(
+   filesDirectory=system.file(
+     "extdata",
+     package="RnaSeqTutorial"),
```

```
+   pattern="[A,C,T,G]{6}\\.bam$",
+   readLength=30L,
+   chr.sel="chr2L",
+   organism="Dmelanogaster",
+   annotationMethod="gff",
+   annotationFile="dmel_synthetic_transcript_r5-54.gff3",
+   count="transcripts",
+   outputFormat="SummarizedExperiment"
+ )
```

That's it. Done much faster than using the 'geneModels' *summarization* argument. This approach is currently being ported to the development version of *easyRNASeq*. There remains a caveat not addressed by this procedure: genes overlapping on the same or opposite strands will still generate double- countings. Adequate warnings would be emitted. A refinement of the method above would be to restrict these genes to their non-overlapping intervals. Commonly the overlaps encompass UTRs and can be safely ignored. In a few case however, it might be difficult to decide wheter to keep or drop them. Note that if the result is returned as an *RNAseq* or *RangedSummarizedExperiment* class object, the overlapping genes are flagged by a boolean. To access this information in an *RNAseq* object, do:

```
> genomicAnnotation(rnaSeq)$overlap
```

and from a *RangedSummarizedExperiment* object:

```
> rowRanges(sumExp)$overlap
```

7.2 Processing a set of human samples

Getting the data If you already have a set of bam files resulting of short read alignments against the human genome, you can just proceed to the next paragraph. If not here is an example on how to retrieve and align data in R. Note that the *Rsubread* is only supported on the linux platform. You'll need to use your aligner of choice and generate BAM files if you're using another platform. However you should still be able to download the SRA/ENA files. The files listed in this example are from the [18] study, but where simply selected for their small size. Still downloading and creating all the necessary file requires times and a computer with a sufficient amount of memory to index the human genome.

```
> library(BSgenome.Hsapiens.UCSC.hg19)
> library(GEOquery)
> library(SRAdb)
> library(Rsamtools)
> library(Rsubread)
> ## create a temp dir
> dir.create(file.path(getwd(),"tmp1234"))
> ## change the working directory
> setwd(file.path(getwd(),"tmp1234"))
> ## init SRA
> sqlfile <- getSRAdbFile()
> ## init a connection
> sra_con <- dbConnect(SQLite(),sqlfile)
> ## list the files
> acc <- c("SRR490224","SRR490225")
> getFASTQinfo( in_acc = acc, srcType = 'ftp' )
> ## get the read files
> getSRAfile( in_acc=acc, sra_con, destDir = getwd(),
```

```
+            fileType = 'fastq', srcType = 'ftp' )
> ## close the connection
> dbDisconnect( sra_con )
> ## write the human genome sequences
> writeXStringSet(Reduce(append,
+                        lapply(seqnames(Hsapiens),
+                               function(nam){
+                                 dss <- DNAStringSet(unmasked(Hsapiens[[nam]]))
+                                 names(dss) <- nam
+                                 dss
+                               })),
+                 file="hg19.fa")
> ## create the indexes
> dir.create("indexes")
> buildindex(basename=file.path("indexes","hg19"),
+            reference="hg19.fa")
> ## align the reads
> sapply(dir(pattern="*\\.gz$"),function(fil){
+   ## decompress the files
+   gunzip(fil)
+
+   ## align
+   align(index=file.path("indexes","hg19"),
+         readfile1=sub("\\.gz$","",fil),
+         nsubreads=2,TH1=1,
+         output_file=sub("\\.fastq\\.gz$","\\.sam",fil)
+         )
+
+   ## create bam files
+   asBam(file=sub("\\.fastq\\.gz$","\\.sam",fil),
+         destination=sub("\\.fastq\\.gz$","",fil),
+         indexDestination=TRUE)
+ })
>
```

Note that this has generated a number of files that you should clean up afterwards, i.e. delete the "tmp1234" folder, once you are done with the use case.

Processing the data First, we start by retrieveing the size of the chromosomes. This is an important information for calculating any feature count. Actually, neither the BSgenome nor any related packages are required for *easyRNASeq* to run. As they are the easiest way to access genomic information such as chromosome lengths within the R/Bioconductor framework, they are made available to the easyRNASeq for that purpose. However, providing a simple named vector is sufficient and therefore is easyRNASeq not limited to existing *BSgenome* organisms. The chromosome size is essential for one reason: to provide a complete representation of the data. When counting reads per features, one get counts for these features that have at least one read aligning to them, i.e. every feature having no reads will be missed. One could then either return only those features having counts or returning a value of 0 counts for those that do not. We do not find these solutions satisfying and to ensure that we provide coherent data, we return the counts for every feature present on the chromosomes. For that purpose, the chromosome size is essential as it allows us to define those features on a chromosome that are located between the last feature having a number a count bigger or equal to one and the end of the chromosome - features, which would otherwise

be ignored. It is as well a mean to monitor that the provided annotation is pertinent. As of version 1.3.5, when using the 'bam' format, it is even easier. Setting the 'chr.sizes' argument to **auto** will result in the chromosome sizes to be retrieved from the BAM header. However, the purpose of this use case is to, at least partly, demonstrate the importance of using appropriate annotations and therefore the use of the 'chr.sizes' argument is demonstrated.

```
> library(BSgenome.Hsapiens.UCSC.hg19)
> chr.sizes=seqlengths(Hsapiens)
```

Then, we list the BAM files.

```
> bamfiles=dir(getwd(),pattern="*\\.bam$")
```

We can now run the easyRNASeq function, fetching the annotation using *biomaRt*. As this is time consuming (about 10 minutes from an average network) and since we might want to work on these annotation to avoid double-counting, we first ask the function to return an instance of the *RNAseq* class. Then, using the genomicAnnotation accessor, we extract the retrieved annotation.

```
> rnaSeq <- easyRNASeq(filesDirectory=getwd(),
+                      organism="Hsapiens",
+                      readLength=58L,
+                      annotationMethod="biomaRt",
+                      count="exons",
+                      filenames=bamfiles[1],
+                      outputFormat="RNAseq"
+                      )
> gAnnot <- genomicAnnotation(rnaSeq)
```

As one can see by looking at this object, it contains 434 "chromosomes", most of which are of no interest to us. For that reason, we first filter it and then save it to the disk for later re-use. The ".rda" extension is a synonym of the ".RData" one and identifies a serialized R data file.

```
> gAnnot <- gAnnot[space(gAnnot) %in% paste("chr",c(1:22,"X","Y","M"),sep=""),]
> save(gAnnot,file="gAnnot.rda")
```

Now using the modified, saved annotation, we can get a count table as follows:

```
> countTable <- easyRNASeq(filesDirectory=getwd(),
+                          organism="Hsapiens",
+                          readLength=58L,
+                          annotationMethod="rda",
+                          annotationFile="gAnnot.rda",
+                          count="exons",
+                          filenames=bamfiles[1]
+                          )
```

Another way to retrieve the annotation is to use the *GenomicFeatures* library. *easyRNASeq* does not yet supports that library automatically, but it can take `GRangesList` object as input; objects that are derivatives of the ones returned by the functions of the *GenomicFeatures* package. A few changes needs to be done to the obtained object so that it can be used by the easyRNASeq: first, a metadata element needs to be updated and then the object needs to be converted into a `GRangesList`.

```
> library(GenomicFeatures)
> hg19.tx <- makeTxDbFromUCSC(genome="hg19", tablename="refGene")
> gAnnot <- exons(hg19.tx)
> colnames(elementMetadata(gAnnot)) <- "exon"
> gAnnot <- split(gAnnot,seqnames(gAnnot))
```

As previously, this annotation can either be saved to disk or be used directly, using the `annotationMethod` "env" and `annotationObject` arguments. Additionaly, one can select chromosome of interest using the `chr.sel` argument. It accepts a vector of chromosome names. In the following, we subset for the chromosome "chr1" only.

```
> countTable <- easyRNASeq(filesDirectory=getwd(),
+                          organism="Hsapiens",
+                          readLength=58L,
+                          annotationMethod="env",
+                          annotationObject=gAnnot,
+                          count="exons",
+                          filenames=bamfiles[1],
+                          chr.sel="chr1"
+                          )
```

Note that in the previous call, we removed the 'chr.sizes' argument, just to demonstrate that the chr.sizes can be retrieved from the bam files. Removing the argument is the same as setting it to its default value: 'auto'

Alternatively, one can use the `outputFormat` argument function to get a *RangedSummarizedExperiment* object back. Mind the comments in section 6, page 24 though.

```
> sumExp <- easyRNASeq(filesDirectory=getwd(),
+                      organism="Hsapiens",
+                      readLength=58L,
+                      annotationMethod="env",
+                      annotationObject=gAnnot,
+                      count="exons",
+                      filenames=bamfiles[1],
+                      chr.sel="chr1",
+                       outputFormat="SummarizedExperiment"
+                      )
```

Finally, instead of returning a count table, we can get a `CountDataSet` instance from the *DESeq* package. This will perform the normalization of the data and generate some Quality Assessment plots. In the present example, it would not yield very sensitive results as we have no replicates (biological). These are important to *DESeq* to accurately model the technical and biological variance. With no replicates for every condition, the dispersion will be based on a "pooled" estimate making the differential expression call lose sensitivity. In addition, *DESeq* is in such cases using a conservative approach (which is good) so you'd get even less significant results. Here, as we have no replicates, we need to pass additional arguments (the last two) that will be tunnelled to the *DESeq* `estimateDispersions` function. See the *DESeq* vignette for more details on these.

```
> conditions <- c("A","B")
> names(conditions) <- bamfiles
> countDataSet <- easyRNASeq(filesDirectory=getwd(),
+                            organism="Hsapiens",
+                            annotationMethod="env",
+                            annotationObject=gAnnot,
+                            count="exons",
+                            filenames=bamfiles,
+                            chr.sel="chr1",
+                            outputFormat="DESeq",
+                            normalize=TRUE,
```

```
+                           conditions=conditions,
+                           fitType="local",
+                           method="blind"
+                         )
```

Note that as the read length differs between the two files: 58 and 76, the `readLength` argument was removed from the previous function call. The read size is then identified automatically.

7.3 Dealing with annotation inconsistencies

This use case shows how to deal with inconsistent annotations, *e.g.* when the chromosome names present in the aligned file are different from those that can be retrieved from an annotation source such as *biomaRt*. First, we have a look at the data, in this case some Illumina export files. Reading in the data using the *ShortRead* package is quite resource demanding as the whole sequences are loaded in memory. Then we look at the chromosome names. These differs from what we expect - UCSC standards - as they have an additional ".fa" extension.

```
> aln <- readAligned("data",type="SolexaExport",pattern="*.txt.gz")
> gc()
> levels(chromosome(aln))
```

```
[1] "chr1.fa"  "chr10.fa" "..."      "chrY.fa"
```

They are different from what *biomaRt* will return as well: *i.e.* 1:19, X, Y and MT plus others. This triple inconsistency will be a problem for *easyRNASeq*. If there were only two sets of names, using the "custom" chromosome name map by-pass (see the Details section of the "?easyRNASeq" help page) would solve the issue. However, in the present particular case, as we are retrieving the annotation from *biomaRt*, we need to precise the name of the organism, which circumvent the chromosome name mapping by-pass. The solution is to first fetch the annotation, modify it and save it as an R data file. Some of the retrieved annotation are "NT" contigs. There are only a few of them, so instead of filtering them out, we just ignore them.

```
> obj <- fetchAnnotation(new('RNAseq',
+                           organismName="Mmusculus"
+                           ),
+                       method="biomaRt")
> gAnnot <- genomicAnnotation(obj)
> length(grep("NT_",space(gAnnot)))
>
```

```
[1] 1181
```

```
> names(gAnnot) <- paste("chr",names(gAnnot),".fa",sep="")
```

As described previously, see page 29 we can save the annotation to a file or use it directly, using the `AnnotationMethod` "env" argument. In that later case, since we did not process the annotation, numerous warnings concerning possible multiple countings of reads will be raised. Double counting reads is not what ones want, see section 7.1 (page 25) on how to adress that.
Note that *easyRNASeq* does support some chromosome names conversion by default. The list of organism for which this is possible can be listed using the knownOrganisms function.
Before going on, we do some cleanup as some of the objects we have generated take large amounts of memory.

```
> rm(aln,obj)
> gc()
```

As on page 29 we get the chromosome sizes. Again, note that the use of the BSgenome is not mandatory. It's just easy as they are available in Bioconductor. Typing in your own chromosome sizes named vector is as valid.

```
> library(BSgenome.Mmusculus.UCSC.mm9)
> chr.sizes<- seqlengths(Mmusculus)
```

We can now create the chromosome name mapping.

```
> chr.map <- data.frame(
+                       from=paste("chr",c(1:19,"X","Y"),".fa",sep=""),
+                       to=paste("chr",c(1:19,"X","Y"),sep=""))
```

Using this chromosome map, we can now summarize the reads per feature of interest. Here we want to look for gene models, so we set the `count` and `summarization` arguments. Note again that the `summarization` argument will be deprecated in the near future and its values merged with the `count` ones. The approach described in section 7.1 (page 25) is a faster way to achieve the same.
Asking to get back an *RNAseq* instance allows us to look at the gene models defined by *easyRNASeq*. This offers the possibility to clean them to avoid multiple counting.
As we have Illumina export data at hand, we need to define a set of Filter to ensure that the data is read properly. Indeed, the export file contains all the reads, so the one that do not pass the chastity filter have to be removed. In addition, some of the other reads are for internal QC, and they have no position. For that reason, we need to filter those too out.
Reading in export data is more resource exhausting that reading in bam files, as we are loading in the sequence and quality information as well, whereas we do not need them.
We will now look through three different set of parameters that will stepwise reduce the number of warnings emitted. These warnings are there to help you understand the different pitfalls that the `easyRNASeq` helps you avoiding when analysing your RNA-Seq data. The first approach generates a lot of warnings, because of the differing annotations, since we are using the entire set of annotation we got. As we do not want to generate all these warnings, these code lines are not evaluated. To get a feel about these warnings, they will look as the follows:

```
Warning messages:
1: In .convertToUCSC(names(genomicAnnotation(obj)), organismName(obj),  :
Your custom map does not define a mapping for the following
chromosome names: chrMT.fa
2: In easyRNASeq(filesDirectory = headDir, organism = "custom", chr.map = chr.map,  :
There are 6096 synthetic exons as determined from your annotation that overlap!
This implies that some reads will be counted more than once!
Is that really what you want?
```

Let us start with the full set of annotation.

```
> rnaSeq <- easyRNASeq(filesDirectory="data",
+                      organism="custom",
+                      chr.map=chr.map,
+                      chr.sizes=chr.sizes,
+                      filter=compose(
+                        naPositionFilter(),
+                        chastityFilter()),
+                      readLength=50L,
+                      annotationMethod="env",
+                      annotationObject=gAnnot,
+                      format="aln",
+                      count="genes",
```

```
+                     summarization= "geneModels",
+                     filenames="1-Feb_ATCACG_L003_R1_001_export.txt.gz",
+                     outputFormat="RNAseq",
+                     nbCore=2
+                     )
```

To reduce the number of warnings emitted, we can select for the chromosome we are interested in as previously done on page 30.

```
> rnaSeq <- easyRNASeq(filesDirectory="data",
+                     organism="custom",
+                     chr.map=chr.map,
+                     chr.sizes=chr.sizes,
+                     chr.sel=chr.map$from,
+                     filter=compose(
+                       naPositionFilter(),
+                       chastityFilter()),
+                     readLength=50L,
+                     annotationMethod="env",
+                     annotationObject=gAnnot,
+                     format="aln",
+                     count="genes",
+                     summarization= "geneModels",
+                     filenames="1-Feb_ATCACG_L003_R1_001_export.txt.gz",
+                     outputFormat="RNAseq",
+                     nbCore=2
+                     )
```

Finally, to further reduce the warnings, we can manipulate the RangedData object to remove the unnecessary annotation.

```
> sel <- grep("NT_",names(gAnnot))
> gAnnot <- RangedData(ranges=ranges(gAnnot)[-sel,],values=values(gAnnot)[-sel,])
> colnames(gAnnot) <- gsub("values\\.","",colnames(gAnnot))
```

This last call will only generate two warnings, one that could be easily dealt with (a complaint about the chrMT). The other one is about double counting and it requires to adapt the annotation.

```
> rnaSeq <- easyRNASeq(filesDirectory="data",
+                     organism="custom",
+                     chr.map=chr.map,
+                     chr.sizes=chr.sizes,
+                     chr.sel=chr.map$from,
+                     filter=compose(
+                       naPositionFilter(),
+                       chastityFilter()),
+                     readLength=50L,
+                     annotationMethod="env",
+                     annotationObject=gAnnot,
+                     format="aln",
+                     count="genes",
+                     summarization= "geneModels",
+                     filenames="1-Feb_ATCACG_L003_R1_001_export.txt.gz",
+                     outputFormat="RNAseq",
```

```
+                       nbCore=2
+                       )
```

8 FAQ

warning:

8.1 Does easyRNASeq support reads of variable length?

Yes it does. The base-pair coverage extracted from the read mapping is actually weighted by the read length to determine better read count estimates.

8.2 Does the easyRNASeq support paired-end reads?

No it does not look at the pair information; hence read counts for PE data will be on average $2x$ larger than those returned from other methods. In the context of DE, this is of little significance though.

9 Session Information

The version number of R[19] and Bioconductor [1] packages loaded for generating the vignette were:

```
R version 3.2.2 (2015-08-14)
Platform: x86_64-apple-darwin13.4.0 (64-bit)
Running under: OS X 10.11.2 (El Capitan)

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats4    parallel  stats     graphics  grDevices utils     datasets  methods
[9] base

other attached packages:
 [1] BSgenome.Dmelanogaster.UCSC.dm3_1.4.0 BSgenome_1.38.0
 [3] rtracklayer_1.30.1                    Biostrings_2.38.2
 [5] XVector_0.10.0                        GenomicRanges_1.22.1
 [7] GenomeInfoDb_1.6.1                    IRanges_2.4.4
 [9] S4Vectors_0.8.3                       BiocGenerics_0.16.1
[11] RnaSeqTutorial_0.8.0                  curl_0.9.4
[13] BiocStyle_1.8.0                       easyRNASeq_2.6.1

loaded via a namespace (and not attached):
 [1] SummarizedExperiment_1.0.1 genefilter_1.52.0          locfit_1.5-9.1
 [4] splines_3.2.2              lattice_0.20-33            htmltools_0.2.6
 [7] yaml_2.1.13                survival_2.38-3            XML_3.98-1.3
[10] DBI_0.3.1                  BiocParallel_1.4.0         RColorBrewer_1.1-2
[13] lambda.r_1.1.7             stringr_1.0.0              zlibbioc_1.16.0
[16] futile.logger_1.4.1        hwriter_1.3.2              devtools_1.9.1
[19] memoise_0.2.1              evaluate_0.8               latticeExtra_0.6-26
[22] Biobase_2.30.0             knitr_1.11                 geneplotter_1.48.0
[25] biomaRt_2.26.1             BiocInstaller_1.20.1       AnnotationDbi_1.32.0
[28] xtable_1.8-0               edgeR_3.12.0               formatR_1.2.1
[31] limma_3.26.3               annotate_1.48.0            ShortRead_1.28.0
[34] Rsamtools_1.22.0           digest_0.6.8               stringi_1.0-1
[37] DESeq_1.22.0               grid_3.2.2                 tools_3.2.2
[40] bitops_1.0-6               magrittr_1.5               RCurl_1.95-4.7
[43] LSD_3.0                    RSQLite_1.0.0              futile.options_1.0.0
[46] rmarkdown_0.8.1            GenomicAlignments_1.6.1    genomeIntervals_1.26.0
[49] intervals_0.15.1
```

10 Final remarks

RNA-seq is still maturating and a lot of new developments are to be expected. If you have any questions, comments, feel free to contact me: nicolas.delhomme *at* umu *dot* se.

References

[1] Robert C Gentleman et al. Bioconductor: open software development for computational biology and bioinformatics. *Genome Biology 2010 11:202*, 5(10):R80, Jan 2004.

[2] Nicolas Delhomme, Ismaël Padioleau, Eileen E Furlong, and Larsm Steinmetz. easyrnaseq: a bioconductor package for processing rna-seq data. *Bioinformatics*, Jul 2012. doi:10.1093/bioinformatics/bts477.

[3] Ali Mortazavi et al. Mapping and quantifying mammalian transcriptomes by rna-seq. *Nature Methods*, 5(7):621–8, Jul 2008.

[4] Charlotte Soneson and Mauro Delorenzi. A comparison of methods for differential expression analysis of rna-seq data. *BMC Bioinformatics*, 14:91, Jan 2013. doi:10.1186/1471-2105-14-91.

[5] Susan Tweedie et al. Flybase: enhancing drosophila gene ontology annotations. *Nucleic Acids Research*, 37(Database issue):D555–9, Jan 2009.

[6] Yang Liao, Gordon K Smyth, and Wei Shi. featurecounts: an efficient general purpose program for assigning sequence reads to genomic features. *Bioinformatics*, Nov 2013. doi:10.1093/bioinformatics/btt656.

[7] Martin Morgan et al. Shortread: a bioconductor package for input, quality assessment and exploration of high-throughput sequence data. *Bioinformatics*, 25(19):2607–8, Oct 2009.

[8] Heng Li et al. The sequence alignment/map format and samtools. *Bioinformatics*, 25(16):2078–9, Aug 2009.

[9] Steffen Durinck et al. Biomart and bioconductor: a powerful link between biological databases and microarray data analysis. *Bioinformatics*, 21(16):3439–40, Aug 2005.

[10] Paul Flicek et al. Ensembl 2011. *Nucleic Acids Research*, 39(Database issue):D800–6, Jan 2011.

[11] Simon Anders and Wolfgang Huber. Differential expression analysis for sequence count data. *Genome Biology 2010 11:202*, 11(10):R106, Oct 2010.

[12] Mark D Robinson et al. edger: a bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*, 26(1):139–40, Jan 2010.

[13] C Trapnell et al. Tophat: discovering splice junctions with rna-seq. *Bioinformatics*, 25(9):1105–1111, May 2009.

[14] Philippe Lefrançois et al. Efficient yeast chip-seq using multiplex short-read dna sequencing. *BMC Genomics*, 10:37, Jan 2009.

[15] Andrew M Smith, Lawrence E Heisler, Robert P St Onge, Eveline Farias-Hesson, Iain M Wallace, John Bodeau, Adam N Harris, Kathleen M Perry, Guri Giaever, Nader Pourmand, and Corey Nislow. Highly-multiplexed barcode sequencing: an efficient method for parallel analysis of pooled samples. *Nucleic Acids Research*, May 2010.

[16] RW Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2):147–160, 1950.

[17] CD Pilcher et al. Inferring hiv transmission dynamics from phylogenetic sequence relationships. *PLoS Med*, 5(3):e69, 2008.

[18] Kersti Jääger, Saiful Islam, Pawel Zajac, Sten Linnarsson, and Toomas Neuman. Rna-seq analysis reveals different dynamics of differentiation of human dermis- and adipose-derived stromal stem cells. *PLoS ONE*, 7(6):e38833, Jan 2012.

[19] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. ISBN 3-900051-07-0. URL: http://www.R-project.org.