rCube - RNA-Rates in R

Leonhard Wachutka[1,*], Carina Demel[2], Julien Gagneur[1]

[1] Department of Informatics, Technical University of Munich, Munich, Germany

[2] Max Planck Institute for biophysical Chemistry, Göttingen, Germany

[*] wachutka (at) in.tum.de

August 4, 2017

Abstract

rCube provides a framework for the estimation of RNA metabolism Rates in R (R^3). The rCube package complements the recently published transient transcriptome sequencing (TT-seq) protocol. It has been shown, that 4sU-labeling and subsequent purification of RNA allows to monitor local RNA synthesis. Therefore, the information from TT-seq/4sU-seq and total RNA-seq samples is used to model RNA synthesis, splicing, and degradation rates based on first-order kinetics. The rCube package works on count data and provides a series of functionalities to extract them from the desired features. It allows to extract junctions and constitutive exons from feature annotations, count reads from BAM-files, and normalize different samples against each other using a variety of different methods.

rCube version: 1.1.0

If you use rCube in published research, please cite:

B. Schwalb, M. Michel, B. Zacher, K. Frühauf, C. Demel, A. Tresch, J. Gageur, and P. Cramer:
TT-seq maps the human transient transcriptome.
Science (2016). doi:10.1126/science.aad9841 [1]

Contents

1 Background

As described in [2], 4sU-seq allows to monitor changes in the RNA metabolism. If cells are exposed to 4sU, they rapidly take up this Uridine analog and incorporate it into newly-synthesized RNAs. This way, newly-synthesized RNAs are labeled and can be extracted from the total RNA in the sample. The longer the labeling time, e.i. the time from 4sU addition to harvesting the cells, the bigger is the proportion of labeled RNAs among all RNAs.

Variations in read counts among different samples can have multiple reasons. One source of variation is the sequencing depth. Even replicates from the same experimental condition may exhibit different read counts based on how deep the samples were sequenced. These variations (up to biological variations) can be overcome by normalization to sequencing depth. In a typical RNA-seq experiment, one wants to compare different samples under different experimental conditions. After extracting the RNA from the cells, the same starting material is used for the library prepartion. Therefore the information is lost, if cells from different samples were expressing different amounts of RNA (both for individual genes or on a global scale). Artificial RNA spike-in sequences can be used to adjust for global sequencing variations between samples. Adding the same volumes of spike-ins to a defined number of cells can help to resolve this problem, as they are subject to the same technical biases than natural RNAs, but their read counts should not be influenced by biological processes. In the case of TT-seq/4sU-seq, we additionally want to rescale 4sU-labeled and total RNA-seq samples, so that the ratio of labeled RNA to total RNA read counts reflects the ratio of labeled RNA to total RNA amounts in the cell. This can be achieved by labeling some of the spike-ins with 4sU during the *in vitro* transcription. Then it is also possible to quantify the amount of unlabeled spike-ins (RNAs) that is not lost during labeled RNA purification, the so-called cross-contamination.

explain time series vs only label total

The normalization based on artifical spike-ins and subsequent estimation of synthesis and degradation rates has been successfully implemented and applied in different studies: In human K562 cells, we investigated synthesis rates and half-lives of different RNA species under steady-state conditions [1]. In another study, we investigated the change of RNA synthesis immediately after T-cell stimulation [3].The sensitivity of TT-seq allowed us to monitor rapid changes in transcription from enhancers and promoters during the immediate response of T cells to ionomycin and phorbol 12-myristate 13-acetate (PMA).

define synthesis, splicing, decay rate

define the read classifications: junction reads E-E, E-I, I-E junction

2 Getting started

This vignette provides a pipeline how to... starting from BAM files... You will learn how to estimate sample specific sequencing depths and cross-contamination rates from spike-in counts. These values can be used to normalize gene expression values obtained by RNA-Seq and thus estimate gene-specific synthesis and degradation rates. By extracting reads spanning junctions, splicing times can be estimated. For more robust estimation, multiple samples with different labeling times are taken into account. Before starting, the package must be loaded by:

```
library("rCube")
```

2.1 Example Data

The `inst/extdata` of the *rCube* package provides two example data sets that should illustrate the two different functionalities of rCube:

The first example data set, "Jurkat", contains bam files from resting and activated Jurkat T-cells for TT-seq and RNA-seq samples. The bamfiles are restricted to the FOS gene (chr14:75278000-75283000) and the artifical spike-ins, subsampled to reduce file size. The full data sets are published in [3]. This example data is used to demonstrate the spike-in normalization method, and the estimation of synthesis and degradation rates for individual 4sU-labeled (TT-seq) and total RNA-seq pairs.

The second data set , ...

3 Conditional synthesis and degradation rates for Jurkat data

Example data sets from a T-cell activation experiment are stored in the `inst/extdata` of the *rCube* package. In this part of the vignette, we will demonstrate

- how reads can be counted for (constitutive) exons and spike-ins,
- how the samples are normalized against each other based on the spike-in read counts,
- how synthesis and degradation rates are obtained for (constitutive) exons
- how gene-specific rates are obtained from exon-specific rates

3.1 Gene model

working on exons/introns/genes... The estimation of synthesis and degradation rates with the *rCube* package relies on read counts. Dependent on the features, for which read counts are provided, the rates can reflect synthesis rates of exons, introns, or full genes. Especially degradation rates may differ between exons and introns. Therefore, the features, which should be used to estimate synthesis and degradation rates, and for which read counts are provided or should be obtained, need to be provided as a *GRanges* object.

Due to numerous transcript isoforms per gene, and the arising problem that for some bases their exonic or intronic nature cannot be unambigously identified, we propose to use the model of constitutive exons/introns from [4]. Hereby, all bases, that belong to an exon/intron in all (annotated) transcript isoforms of the same gene, are thought to be part of "constitutive" exons/introns. In the following, we have an example annotation from the FOS gene (not comprehensive) to illustrate how constitutive exons can be extracted from an exon annotation.

```
data("exampleExons")
exampleExons
```

```
## GRanges object with 7 ranges and 3 metadata columns:
##       seqnames              ranges strand |     type           gene_id
##          <Rle>           <IRanges>  <Rle> | <factor>       <character>
```

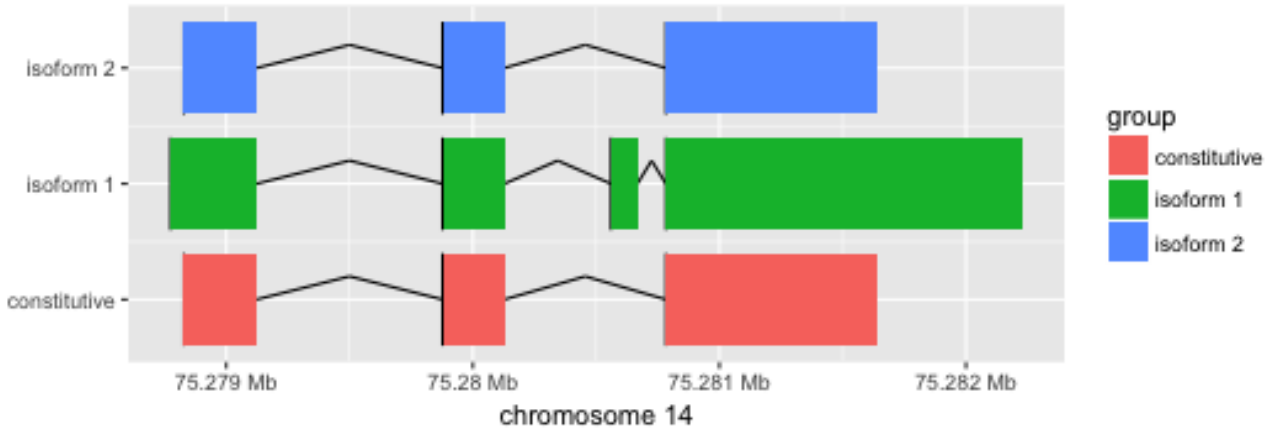

Figure 1: Illustration of two transcript isoforms for the FOS gene and the resulting constitutive exons

```
##   [1]    chr14 [75278774, 75279123]      + |     exon ENSG00000170345.9
##   [2]    chr14 [75278828, 75279123]      + |     exon ENSG00000170345.9
##   [3]    chr14 [75279877, 75280128]      + |     exon ENSG00000170345.9
##   [4]    chr14 [75279877, 75280128]      + |     exon ENSG00000170345.9
##   [5]    chr14 [75280560, 75280667]      + |     exon ENSG00000170345.9
##   [6]    chr14 [75280783, 75282230]      + |     exon ENSG00000170345.9
##   [7]    chr14 [75280783, 75281636]      + |     exon ENSG00000170345.9
##           transcript_id
##             <character>
##   [1] ENST00000303562.8
##   [2] ENST00000535987.5
##   [3] ENST00000303562.8
##   [4] ENST00000535987.5
##   [5] ENST00000303562.8
##   [6] ENST00000303562.8
##   [7] ENST00000535987.5
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

```
constitutiveExons <- createConstitutiveFeaturesGRangesFromGRanges(exampleExons,
                                                                  BPPARAM=NULL,
                                                                  ncores=1)
constitutiveExons
```

```
## GRanges object with 3 ranges and 2 metadata columns:
##           seqnames              ranges strand |                type
##              <Rle>           <IRanges>  <Rle> |            <factor>
##   CF00001    chr14 [75278828, 75279123]      + | constitutive feature
##   CF00002    chr14 [75279877, 75280128]      + | constitutive feature
##   CF00003    chr14 [75280783, 75281636]      + | constitutive feature
##                   gene_id
##                 <character>
##   CF00001 ENSG00000170345.9
##   CF00002 ENSG00000170345.9
##   CF00003 ENSG00000170345.9
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

Please note, for the subsequent workflow it is not necessary to extract constitutive exons. Any kind of *GRanges* object can be used as feature annotation (e.g. full genes, introns, ...).

3.2 Experimental Design

The rCube package works on *rCubeExperiment* containers, that rely on the *SummerizedExperiment* class. Objects of this class are used as input for the whole workflow, starting from read counting, normalization, dispersion estimation, to rate estimation. Most of these steps return an updated and extended *rCubeExperiment* object.

The `rowRanges` of the *rCubeExperiment* is a *GRanges* annotation of features, for which RNA rates should be estimated. Experimental sample information can be either provided by a experimental design matrix or this information can be extracted from the BAM-file names (when they fulfil the required structure).

We first look at the experimental design file `experimentalDesign.txt`, that can be imported as a *data.frame*.

```
folder <- system.file("extdata/Jurkat", package='rCube')
folder
```

```
## [1] "/Library/Frameworks/R.framework/Versions/3.4/Resources/library/rCube/extdata/Jurkat"
```

```
expDesign <- read.delim(file.path(folder, "experimentalDesign.txt"))
expDesign
```

```
##            sample condition LT labelingTime replicate            filename
## 1   Resting_L_5_1   Resting  L            5         1   Resting_L_5_1.bam
## 2   Resting_L_5_2   Resting  L            5         2   Resting_L_5_2.bam
## 3   Resting_T_5_1   Resting  T            5         1   Resting_T_5_1.bam
## 4   Resting_T_5_2   Resting  T            5         2   Resting_T_5_2.bam
## 5 Activated_L_5_1 Activated  L            5         1 Activated_L_5_1.bam
## 6 Activated_L_5_2 Activated  L            5         2 Activated_L_5_2.bam
## 7 Activated_T_5_1 Activated  T            5         1 Activated_T_5_1.bam
## 8 Activated_T_5_2 Activated  T            5         2 Activated_T_5_2.bam
```

Together with the feature annotation, for which we want to estimate synthesis and degradation rates, we can construct the *rCubeExperiment*:

```
exonCounts <- setupExperiment(constitutiveExons, designMatrix=expDesign, files=NULL)
class(exonCounts)
```

```
## [1] "rCubeExperiment"
## attr(,"package")
## [1] "rCube"
```

Alternatively, the experimental design matrix can be constructed from the bam file names internally, if they follow the following convention `{condition}_{L|T}_{labelingTime}_{replicate}.bam`

```
bamfiles <- list.files(folder, pattern="*.bam$", full.names=TRUE)
basename(bamfiles)
```

```
## [1] "ActivatedJurkat_L_5_1.bam" "ActivatedJurkat_L_5_2.bam"
## [3] "ActivatedJurkat_T_5_1.bam" "ActivatedJurkat_T_5_2.bam"
## [5] "RestingJurkat_L_5_1.bam"   "RestingJurkat_L_5_2.bam"
## [7] "RestingJurkat_T_5_1.bam"   "RestingJurkat_T_5_2.bam"
```

```
exonCounts <- setupExperiment(constitutiveExons, designMatrix=NULL, files=bamfiles)
```

The individual information from the *rCubeExperiment* can be assessed by:

```
# feature information
rowRanges(exonCounts)
```

```
# sample information
colData(exonCounts)

# read counts
assay(exonCounts)
```

The resulting *rCubeExperiment* object can now be used to count reads.

3.3 Counting

All RNA rate estimations of this package rely on read counts. These can be either provided as count matrices, or read counts can be obtained from BAM files using the *rCube* pipeline.

For read counting, we use the `readGAlignmentPairs` in a parallel fashion:

```
assay(exonCounts)
```

```
##         ActivatedJurkat_L_5_1 ActivatedJurkat_L_5_2 ActivatedJurkat_T_5_1
## CF00001                    NA                    NA                    NA
## CF00002                    NA                    NA                    NA
## CF00003                    NA                    NA                    NA
##         ActivatedJurkat_T_5_2 RestingJurkat_L_5_1 RestingJurkat_L_5_2
## CF00001                    NA                  NA                  NA
## CF00002                    NA                  NA                  NA
## CF00003                    NA                  NA                  NA
##         RestingJurkat_T_5_1 RestingJurkat_T_5_2
## CF00001                  NA                  NA
## CF00002                  NA                  NA
## CF00003                  NA                  NA
```

```
exonCounts <- countFeatures(exonCounts,
                            scanBamParam=ScanBamParam(flag=scanBamFlag(isSecondaryAlignment=FALSE)),
                            BPPARAM=NULL,
                            verbose=FALSE)

assay(exonCounts)
```

```
##         ActivatedJurkat_L_5_1 ActivatedJurkat_L_5_2 ActivatedJurkat_T_5_1
## CF00001                    74                   123                     6
## CF00002                   112                   195                     4
## CF00003                   481                   610                    17
##         ActivatedJurkat_T_5_2 RestingJurkat_L_5_1 RestingJurkat_L_5_2
## CF00001                     6                   0                   4
## CF00002                     9                   4                   6
## CF00003                    23                  15                  17
##         RestingJurkat_T_5_1 RestingJurkat_T_5_2
## CF00001                   4                   2
## CF00002                   2                   3
## CF00003                   1                   1
```

In case you already have counted reads on your featrues of interest, count matrices can be assigned to the correctly formatted, empty *rCubeExperiment* object.

3.4 Spike-ins

The artifical spike-in annotations and labeling information can be loaded via:

```
data("spikeins")
data("spikeinLabeling")
spikeinLengths <- width(spikeins)
```

3.5 Spike-in design

An empty *rCubeExperiment* for the artificial spike-ins additionally requires information about the length and the labeling status of each spikein, and can be constructed as follows:

```
spikeinCounts <- setupExperimentSpikeins(rows=spikeins,
                                         designMatrix=expDesign,
                                         length=spikeinLengths,
                                         labelingState=spikeinLabeling)
```

3.6 Spike-in counting

```
spikeinCounts <- countSpikeins(spikeinCounts,
                   scanBamParam=ScanBamParam(flag=scanBamFlag(isSecondaryAlignment=FALSE)),
                   BPPARAM=NULL,
                   verbose=FALSE)
assay(spikeinCounts)
```

```
##         Resting_L_5_1 Resting_L_5_2 Resting_T_5_1 Resting_T_5_2 Activated_L_5_1
## Spike2           3648          4416           492           541            4416
## Spike12            75           109           369           408             129
## Spike4           4060          4399           414           456            4031
## Spike5             43            33           472           401             105
## Spike8           4455          5205           397           432            4927
## Spike9             18            13           249           283              66
##         Activated_L_5_2 Activated_T_5_1 Activated_T_5_2
## Spike2             3791             695             498
## Spike12              93             496             349
## Spike4             3800             548             425
## Spike5               30             573             363
## Spike8             4272             582             412
## Spike9               14             333             187
```

The distribution of 4sU-labeled and unlabeld spike-ins among different samples can be illustrated by the function `plotSpikeinCountsVsSample`. Figure 2 shows the read counts of spike-ins in the Jurkat example data set.

```
plotSpikeinCountsVsSample(spikeinCounts)
```

Naturally, labeled spike-ins (Spike2, Spike4, Spike8) should be enriched in labeled samples ("L"), whereas unlabeled spike-ins (Spike5, Spike9, Spike12) should be depleted from these samples. In total RNA-seq samples ("T"), all spike-ins should be present to a similar extend.

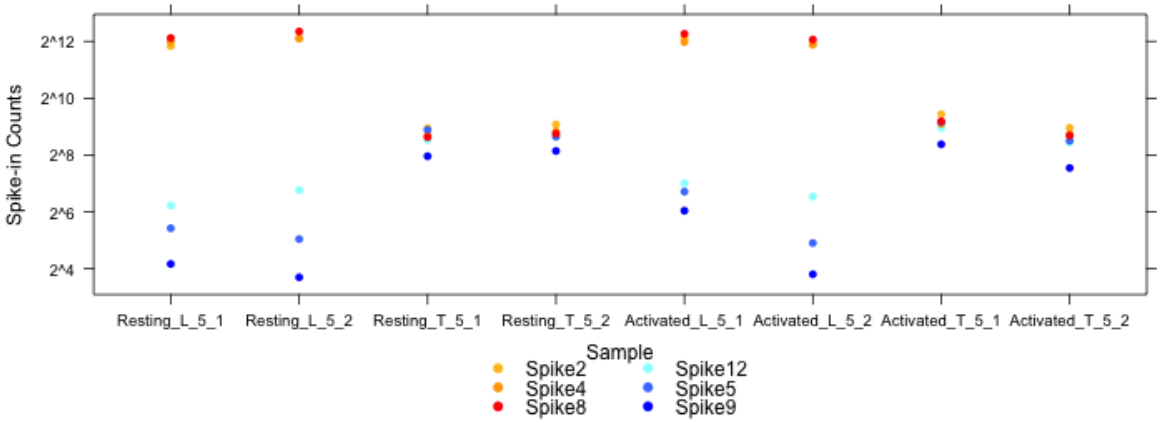

Figure 2: **Spike-in read counts in different samples.** The code that creates this figure is shown in the code chunk.

3.7 Size factor based on spike-ins

We provide two different normalization schemes. In the experimental setup with multiple conditions, the sample specific parameters like sequencing depth and cross-contamination rate are estimated from spike-in read counts only. Therefore, we fit a generalized linear model (GLM) of the Negative Binomial family with a log link function. The response of the GLM are the observed spike-in counts, and the terms that specify the linear predictor of the response are comprised of:

- a sample specific factor (that reflects the sample specific sequencing depth),
- a labeled sample specific factor (that reflects the control for cross contamination (only estimated for unlabeled spike-ins in labeled samples)), and
- a spike-in specific factor to allow for some spike-in specific variation e.g. due to sequence biases.

Additionally, the length of each spike-in is used as an offset, i.e. a known slope for the covariate.

```
exonCounts <- estimateSizeFactors(exonCounts, spikeinCounts, method="spikeinGLM")
colnames(colData(exonCounts))
```

```
## [1] "filename"           "sample"             "condition"
## [4] "LT"                 "labelingTime"       "replicate"
## [7] "sequencing.depth"   "cross.contamination"
```

```
exonCounts$sequencing.depth
```

```
##   Resting_L_5_1   Resting_L_5_2   Resting_T_5_1   Resting_T_5_2 Activated_L_5_1
##          0.9121          1.0495          0.1024          0.1079          1.0000
## Activated_L_5_2 Activated_T_5_1 Activated_T_5_2
##          0.8884          0.1376          0.0935
```

```
exonCounts$cross.contamination
```

```
##   Resting_L_5_1   Resting_L_5_2   Resting_T_5_1   Resting_T_5_2 Activated_L_5_1
##          0.0137          0.0130          1.0000          1.0000          0.0288
## Activated_L_5_2 Activated_T_5_1 Activated_T_5_2
##          0.0138          1.0000          1.0000
```

Note, the cross-contamination value for all total RNA-seq samples is 1, as 100% of the unlabeled RNAs are supposed to be in the sample. Additional fitting results are stored in the `metadata` of the resulting *rCubeExperiment* object.

```
metadata(exonCounts)
```

3.8 Providing gene-wise dispersion estimates

Usually, read counts in different RNA-seq samples underly fluctuations due to biological or technical variances. To take these fluctuations into account, we estimate each gene's dispersion. For each gene, a single dispersion estimate for all 4sU-Seq samples and for all Total RNA-Seq samples is needed. The wrapper function `estimateSizeDispersions` offers different methods to estimate a gene's dispersion Here, we can use the method provided in the DESeq2 package [5]. The DESeq algorithm is applied all genes, while separating the count table according to the RNA-Seq protocol (labeled or total RNA). It is possible to choose between all provided DESeq dispersion estimates, namely the genewise maximum likelihood dispersion estimate ("dispGeneEst"), the smooth curve fitted through the gene-wise disperion estimates ("dispFit") and the genewise dispersion estimates shrunken towards the fitted curve ("dispMAP", default). The input is an *rCubeExperiment* object with read counts for the features of interest. The function returns an updated *rCubeExperiment* object with two additional columns in the `rowRanges`, namely `dispersion_L` and `dispersion_T`.

```
exonCounts <- estimateSizeDispersions(exonCounts, method='DESeqDispGeneEst')
rowRanges(exonCounts)
```

```
## GRanges object with 3 ranges and 3 metadata columns:
##           seqnames              ranges strand |        group dispersion_L
##              <Rle>           <IRanges>  <Rle> |     <factor>    <numeric>
##   CF00001    chr14 [75278828, 75279123]      + | constitutive        1e-08
##   CF00002    chr14 [75279877, 75280128]      + | constitutive        1e-08
##   CF00003    chr14 [75280783, 75281636]      + | constitutive        1e-08
##           dispersion_T
##              <numeric>
##   CF00001        1e-08
##   CF00002        1e-08
##   CF00003        1e-08
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

3.9 Feature-specific synthesis and degratation rate estimates

After estimating sequencing depth and cross-contamination rates per sample (see Section 3.7) and extracting feature-specific dispersion estimates (see Section 3.8), we can now estimate RNA synthesis and degradation rate for each feature and condition individually. Multiple replicates for the same condition can be used for a joint estimation. The user has to specify for which replicate or combination of replicates the results should be estimated. Therefore, the `replicate` parameter is a vector of all combinations that should be evaluated. For the joint estimation for multiple replicates, these have to be given as a string separeted by a ":". In the following example, we will obtain individual results for replicate 1 and 2 and also results for a joint estimation.

```
rates <- estimateRateByFirstOrderKinetics(exonCounts,
                                          replicate=c(1, 2, "1:2"),
                                          method='single',
                                          BPPARAM=BiocParallel::MulticoreParam(1))
```

3.10 describe the different fitting functions

3.11 describe the class of the returned object (rCubeRates)

4 Labeling time series

4.1 Experimental Design

4.2 Read classification

4.3 Gene model

(exon, into and junctions) by gff de novo + gff

4.4 Counting

4.5 Spike-ins

4.6 Spike-in design

4.7 Spike-in counting

4.8 Size factor based on spike-in

4.9 Estimate dispersion

4.10 Fit the rates

4.11 describe the different fitting functions

4.12 describe the class of the returned object (rCubeRates)

5 Session Information

This vignette was generated using the following package versions:

```
sessionInfo()
```

```
## R version 3.4.1 (2017-06-30)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: OS X El Capitan 10.11.6
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.dylib
##
## 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] parallel  stats4    stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] Rsamtools_1.29.0           Biostrings_2.45.3
##  [3] XVector_0.17.0             ggbio_1.25.3
##  [5] ggplot2_2.2.1              rCube_1.1.0
```

```
##  [7] SummarizedExperiment_1.7.5 DelayedArray_0.3.19
##  [9] matrixStats_0.52.2         Biobase_2.37.2
## [11] GenomicRanges_1.29.12      GenomeInfoDb_1.13.4
## [13] IRanges_2.11.12            S4Vectors_0.15.5
## [15] BiocGenerics_0.23.0        knitr_1.16
##
## loaded via a namespace (and not attached):
##  [1] ProtGenerics_1.9.0            bitops_1.0-6
##  [3] bit64_0.9-7                   RColorBrewer_1.1-2
##  [5] progress_1.1.2                httr_1.2.1
##  [7] rprojroot_1.2                 tools_3.4.1
##  [9] backports_1.1.0               R6_2.2.2
## [11] rpart_4.1-11                  Hmisc_4.0-3
## [13] DBI_0.7                       lazyeval_0.2.0
## [15] colorspace_1.3-2              nnet_7.3-12
## [17] gridExtra_2.2.1               prettyunits_1.0.2
## [19] GGally_1.3.2                  DESeq2_1.17.13
## [21] curl_2.8.1                    bit_1.1-12
## [23] compiler_3.4.1                graph_1.55.0
## [25] htmlTable_1.9                 rtracklayer_1.37.3
## [27] scales_0.4.1                  checkmate_1.8.3
## [29] genefilter_1.59.0             RBGL_1.53.0
## [31] stringr_1.2.0                 digest_0.6.12
## [33] foreign_0.8-69                rmarkdown_1.6
## [35] base64enc_0.1-3               dichromat_2.0-0
## [37] htmltools_0.3.6               ensembldb_2.1.10
## [39] BSgenome_1.45.1               highr_0.6
## [41] htmlwidgets_0.9               rlang_0.1.1
## [43] RSQLite_2.0                   BiocInstaller_1.27.2
## [45] shiny_1.0.3                   BiocParallel_1.11.5
## [47] acepack_1.4.1                 VariantAnnotation_1.23.6
## [49] RCurl_1.95-4.8                magrittr_1.5
## [51] GenomeInfoDbData_0.99.1       Formula_1.2-2
## [53] Matrix_1.2-10                 Rcpp_0.12.12
## [55] munsell_0.4.3                 stringi_1.1.5
## [57] yaml_2.1.14                   MASS_7.3-47
## [59] zlibbioc_1.23.0               plyr_1.8.4
## [61] AnnotationHub_2.9.5           grid_3.4.1
## [63] blob_1.1.0                    lattice_0.20-35
## [65] splines_3.4.1                 GenomicFeatures_1.29.8
## [67] annotate_1.55.0               locfit_1.5-9.1
## [69] geneplotter_1.55.0            reshape2_1.4.2
## [71] biomaRt_2.33.4                XML_3.98-1.9
## [73] evaluate_0.10.1               biovizBase_1.25.1
## [75] latticeExtra_0.6-28           data.table_1.10.4
## [77] httpuv_1.3.5                  gtable_0.2.0
## [79] reshape_0.8.6                 assertthat_0.2.0
## [81] mime_0.5                      xtable_1.8-2
## [83] AnnotationFilter_1.1.3        survival_2.41-3
## [85] OrganismDbi_1.19.0            tibble_1.3.3
## [87] GenomicAlignments_1.13.4      AnnotationDbi_1.39.2
## [89] memoise_1.1.0                 cluster_2.0.6
## [91] interactiveDisplayBase_1.15.0 BiocStyle_2.5.8
```

6 References

[1] Björn Schwalb, Margaux Michel, Benedikt Zacher, Carina Frühauf, Katja Demel, Achim Tresch, Julien Gagneur, and Patrick Cramer. TT-seq maps the human transient transcriptome. *Science*, 352(6290):1225–1228, 2016.

[2] Christian Miller, Björn Schwalb, Kerstin Maier, Daniel Schulz, Sebastian Dümcke, Benedikt Zacher, Andreas Mayer, Jasmin Sydow, Lisa Marcinowski, Lars Dölken, Dietmar E Martin, Achim Tresch, and Patrick Cramer. Dynamic transcriptome analysis measures rates of mRNA synthesis and decay in yeast. *Molecular Systems Biology*, 7(458):458, jan 2011. URL: http://www.nature.com/doifinder/10.1038/msb.2010.112http://dx.doi.org/10.1038/msb.2010.112, doi:10.1038/msb.2010.112.

[3] Margaux Michel, Carina Demel, Benedikt Zacher, Björn Schwalb, Stefan Krebs, Julien Gagneur, and Patrick Cramer. TT-seq captures enhancer landscapes immediately after T-cell stimulation. *Molecular Systems Biology*, 13(3):920, 2017. doi:10.15252/msb.20167507.

[4] James H Bullard, Elizabeth Purdom, Kasper D Hansen, and Sandrine Dudoit. Evaluation of statistical methods for normalization and differential expression in mRNA-Seq experiments. *BMC Bioinformatics*, 11:94, 2010. doi:10.1186/1471-2105-11-94.

[5] Michael I Love, Wolfgang Huber, and Simon Anders. Moderated estimation of fold change and dispersion for RNA-Seq data with DESeq2. *Genome Biology*, 15(12):550, 2014.