AllelicImbalance Vignette

Jesper R. Gadin and Lasse Folkersen

2018-04-30

Contents

#Introduction This *AllelicImbalance* package contains functions for investigating allelic imbalance effects in RNA-seq data. Maternal and paternal alleles could be expected to show identical transcription rate, resulting in a 50%-50% mix of maternal and paternal mRNA in a sample. However, this turns out to sometimes not be the case. The most extreme example is the X-chromosome inactivation in females, but many autosomal genes also have deviations from identical transcription rate. The causes of this are not always known, but one likely cause is the difference in DNA, namely heterozygous SNPs, affecting enhancers, promoter regions, splicing and stability. Identifying this allelic imbalance is therefore of interest to the characterization of the genome and the aim of the AllelicImbalance package is to facilitate this.

Load AllelicImbalance

```
library(AllelicImbalance)
```

1 ASEset

The ASEset object is the central class of objects in the AllelicImbalance package. The ASEset object has the RangedSummarizedExperiment from the *SummarizedExperiment* package as parent class, and all functions you can apply on this class you can also apply on an ASEset.

1.1 Simple example of building an ASEset object

In this section we will walk through the various ways an ASEset object can be created. Although the preprocessing of RNA-seq data is not the primary focus of this package, it is a necessary step before analysis. There exists several different methods for obtaining a bam file, and this section should just be considered an example. For further details we refer to the web-pages of tophat, bowtie, bwa and samtools found in the links section at the end of this document.

```
wget ftp://ftp.sra.ebi.ac.uk/vol1/fastq/ERR009/ERR009135/*
bowtie -q --best --threads 5 --sam hg19 +
>      -1 ERR009135_1.fastq.gz -2 ERR009135_2.fastq.gz "ERR009135.sam"
samtools view -S -b ERR009135.sam > ERR009135.bam
```

In the above code one paired-end RNA sequencing sample is downloaded and aligned to the human genome, then converted to bam using samtools. The resulting bam files can be the direct input to the AllelicImbalance package. Other aligners can be used as well, as long as bam files are provided as input. The example code following illustrates how to use the import mechanism on a chromosome 17-located subset of 20 RNA-seq experiments of HapMap samples. The output is an ASEset object containing allele counts for all heterozygote coding SNPs in the region.

```
searchArea <- GRanges(seqnames = c("17"), ranges = IRanges(79478301, 79478361))
pathToFiles <- system.file("extdata/ERP000101_subset",
                                package = "AllelicImbalance")
reads <- impBamGAL(pathToFiles, searchArea, verbose = FALSE)
heterozygotePositions <- scanForHeterozygotes(reads, verbose = FALSE)
countList <- getAlleleCounts(reads, heterozygotePositions, verbose = FALSE)
```

```
a.simple <- ASEsetFromCountList(heterozygotePositions, countList)
a.simple
## class: ASEset
## dim: 3 20
## metadata(0):
## assays(3): countsPlus mapBias phase
## rownames(3): chr17_79478287 chr17_79478331 chr17_79478334
## rowData names(0):
## colnames(20): ERR009097.bam ERR009102.bam ... ERR009160.bam
##   ERR009167.bam
## colData names(0):
```

1.2 Building an ASEset object using Bcf or Vcf files

If more than a few genes and a few samples are analyzed we recommend that a SNP-call is instead made using the samtools mpileup function (see links section). The scanForHeterozygotes function is merely a simple SNP-caller and it is not as computationally optimized as e.g. mpileup. In this bash code we download reference sequence for chromosome 17 and show how to generate mpileup calls on one of the HapMap samples that were described above.

```
#download the reference chromosome in fasta format
wget ftp://hgdownload.cse.ucsc.edu/goldenPath/hg19/chromosomes/chr17.fa.gz

#Vcf format
samtools mpileup -uf hg19.fa ERR009135.bam > ERR009135.vcf

#Bcf format
samtools mpileup --BCF --fasta-ref hg19.fa \
    --output ERR009135.bcf ERR009135.bam
```

Samtools mpileup generates by default a Vcf file which contains SNP and short INDEL positions. Piping the output to bcftools we get its binary equivalent (Bcf), which takes less space and can be queried more effective.

The Vcf file is a text file that stores information about positions in the genome. In addition to the location, stored informationn could for example be genotype, phase, reference and alternative alleles for a collection of samples. More detailed information can be found following this link.

In the *VariantAnnotation* package there is a lot of useful tools to handle Vcf files. The readVcf function reads Vcf data into R, which can be subset to only ranges by granges to get a GRanges object that is the object type required by the getAlleleCounts function.

```
library(VariantAnnotation)
##
## Attaching package: 'VariantAnnotation'
## The following object is masked from 'package:base':
##
##     tabulate
pathToVcf <- paste(pathToFiles,"/ERP000101.vcf",sep="")
VCF <- readVcf(pathToVcf,"hg19")
```

```
gr <- granges(VCF)

#only use bi-allelic positions
gr.filt <- gr[width(mcols(gr)[,"REF"]) == 1 |
                unlist(lapply(mcols(gr)[,"ALT"],width))==1]

countList <- getAlleleCounts(reads, gr.filt, verbose=FALSE)
a.vcf <- ASEsetFromCountList(rowRanges = gr, countList)
```

With the Bcf files the process of generating an ASEset object starts with a call to the impBcfGR function instead. This function will import the Bcf file containing all SNP calls that were generated with the samtools mpileup function.

```
BcfGR <- impBcfGR(pathToFiles,searchArea,verbose=FALSE)
countListBcf <- getAlleleCounts(reads, BcfGR,verbose=FALSE)
a.bcf <- ASEsetFromCountList(BcfGR, countListBcf)
```

1.3 Using strand information

Many RNA-seq experiments do not yield useful information on the strand from which a given read was made. This is because they involve a step in which a double-stranded cDNA is created without tracking strand-information. Some RNA-seq setups do however give this information and in those cases it is important to keep track of strand in the ASE-experiment. The example data from above is from an experiment which created double-stranded cDNA before labelling and so the '+' and '-' information in it is arbitrary. However, if we assume that the information has strand information, then the correct procedure is as follows:

```
plus <- getAlleleCounts(reads, heterozygotePositions, strand="+",verbose=F)
minus <- getAlleleCounts(reads, heterozygotePositions, strand="-",verbose=F)

a.stranded <-
ASEsetFromCountList(
heterozygotePositions,
countListPlus=plus,
countListMinus=minus
)
a.stranded
## class: ASEset
## dim: 3 20
## metadata(0):
## assays(4): countsPlus countsMinus mapBias phase
## rownames(3): chr17_79478287 chr17_79478331 chr17_79478334
## rowData names(0):
## colnames(20): ERR009097.bam ERR009102.bam ... ERR009160.bam
##   ERR009167.bam
## colData names(0):
```

The main effect of doing this, is in the plotting functions which will separate reads from different strands if they are specified as done here. It is important, however, to make sure that the imported RNA-seq experiment does in fact have proper labeling and tracking of strand information before proceeding with this method.

1.4 Two useful helper functions

At this stage it is worth highlighting two useful helper functions that both uses existing BioC annotation objects. One is the getAreaFromGeneNames which quickly retrieves the above mentioned searchArea when given just genesymbols as input, and relies on *org.Hs.eg.db*. The other other is the getSnpIdFromLocation function which attempts to rename location-based SNP names to established rs-IDs in case they exist. These functions work as follows:

```
#Getting searchArea from genesymbol
library(org.Hs.eg.db )
searchArea<-getAreaFromGeneNames("ACTG1",org.Hs.eg.db)

#Getting rs-IDs
library(SNPlocs.Hsapiens.dbSNP144.GRCh37)
gr <- rowRanges(a.simple)
updatedGRanges<-getSnpIdFromLocation(gr, SNPlocs.Hsapiens.dbSNP144.GRCh37)
rowRanges(a.simple)<-updatedGRanges
```

1.5 Adding phenotype data

Typically an RNA-seq experiment will include additional information about each sample. It is an advantage to include this information when creating an ASEset because it can be used for subsequent highlights or subsetting in plotting and analysis functions.

```
#simulate phenotype data
pdata <- DataFrame(
    Treatment=sample(c("ChIP", "Input"),length(reads),replace=TRUE),
    Gender=sample(c("male", "female"),length(reads),replace=TRUE),
    row.names=paste("individual",1:length(reads),sep=""))

#make new ASEset with pdata
a.new <- ASEsetFromCountList(
        heterozygotePositions,
        countList,
        colData=pdata)

#add to existing object
colData(a.simple) <- pdata
```

1.6 Adding genotype information

The genotype of the coding SNPs can be set by the command `genotype(x) <- value`. If the genotype information is not available it can be inferred from the allele counts. The major allele will then be the allele with most counts and minor with second most counts. The notation is major/minor and for the example G/C, The G allele would be the major and C the minor allele. To be able to infer and store the genotype information it is required to first declare reference alleles. In cases when it is not important to know which allele is reference, the reference allele can be inferred from allele counts, by random takingone of the most expressed alleles.

```
#infer and add genotype require declaration of the reference allele
ref(a.simple) <- randomRef(a.simple)
genotype(a.simple) <- inferGenotypes(a.simple)

#access to genotype information requires not only the information about the
#reference allele but also the alternative allele
alt(a.simple) <- inferAltAllele(a.simple)
genotype(a.simple)[,1:4]
##      [,1]  [,2]  [,3]  [,4]
## [1,] "G/A" "G/A" NA    "G/A"
## [2,] "T/G" "T/G" "T/G" "T/G"
## [3,] "C/G" "C/G" "C/G" "C/G"
```

1.7 Adding phase information

For some functionality phase information is necessary. Phasing can be obtained from many external sources e.g. samtools. The phase information is often present in VCF-files. The lines below show how to access that information and transfer it to an ASEset. The ASEset follows the VCF-conventions on how to describe the phase, i.e. each patients phase will be described by the established notation of the form "1|0","1|1" or "1/0". There the left number is the description of the maternal allele and the right number is the description of the paternal allele. If it is 0 the allele is the same as the reference allele, and if it is 1 it is the alternative allele. For "|" the phase is known and for "/" the phase is not known. Note, that in the AllelicImbalance package only bi-allelic expression is allowed.

The phase can be manually added by constructing a user-generated matrix, or transforming the data into a matrix from an external source. The most convenient way of importing phase information is probably by reading it from a Vcf file, which is commonly used to store phase information. The readGT function from the *VariantAnnotation* package will return a matrix ready to just attach to an ASEset object.

```
#construct an example phase matrix
set.seed(1)
rows <-nrow(a.simple)
cols <- ncol(a.simple)
p1 <- matrix(sample(c(1,0),replace=TRUE, size=rows*cols), nrow=rows, ncol=cols)
p2 <- matrix(sample(c(1,0),replace=TRUE, size=rows*cols), nrow=rows, ncol=cols)
phase.mat <- matrix(paste(p1,sample(c("|","|","/"), size=rows*cols,
                          replace=TRUE), p2, sep=""), nrow=rows, ncol=cols)
```

```
phase(a.simple) <- phase.mat
```

```
#load VariantAnnotation and use the phase information from a Vcf file
pathToVcf <- system.file("extdata/ERP000101_subset/ERP000101.vcf",
                         package = "AllelicImbalance")
p <- readGT(pathToVcf)
#The example Vcf file contains only 19 out of our 20 samples
#So we have to subset and order
a.subset <- a.simple[,colnames(a.simple) %in% colnames(p)]
p.subset <- p[, colnames(p) %in% colnames(a.subset)]
p.ordered <- p.subset[ , match(colnames(a.subset), colnames(p.subset))]
```

1.8 Adding reference and alternative allele information

Having the information of reference and alternative allele is important to investigate any presence of mapping bias. It is also important to be able to use phasing information. The reference and alternative alleles are stored in the meta-columns and can be accessed and set through the mcols() function. All functions that require reference or alternative allele will visit the meta-columns "ref" or "alt" to extract that information.

```
#from simulated data
ref(a.simple) <- c("G","T","C")

#infer and set alternative allele
alt <- inferAltAllele(a.simple)
alt(a.simple) <- alt

#from reference genome
data(ASEset.sim)
fasta <- system.file('extdata/hg19.chr17.subset.fa', package='AllelicImbalance')
ref <- refAllele(ASEset.sim,fasta=fasta)
ref(ASEset.sim) <- ref
```

Using reference allele information when measuring the impact of mapping bias globally, can be done by measuring the average reference allele fraction from a representative set of SNPs. Below the simulated example uses 1000 SNPs to assess any presence of mapping bias. Any deviations from 0.5 suggests a bias favouring one or the other allele. The most likely outcome is a value higher than 0.5 and is probably due to mapping bias, i.e., the reference allele actually has mapped more often than the alternative allele.

```
#make an example countList including a global sample of SNPs
set.seed(1)
countListUnknown <- list()
samples <- paste(rep("sample",10),1:10,sep="")
snps <- 1000
for(snp in 1:snps){
    count<-matrix(0, nrow=length(samples), ncol=4,
                  dimnames=list(samples, c('A','T','G','C')))
    alleles <- sample(1:4, 2)
    for(sample in 1:length(samples)){
```

```
            count[sample, alleles] <- as.integer(rnorm(2,mean=50,sd=10))
        }
        countListUnknown[[snp]] <- count
    }

    #make example rowRanges for the corresponding information
    gr <- GRanges(
        seqnames = Rle(sample(paste("chr",1:22,sep=""),snps, replace=TRUE)),
        ranges = IRanges(1:snps, width = 1, names= paste("snp",1:snps,sep="")),
        strand="*"
    )

    #make ASEset
    a <- ASEsetFromCountList(gr, countListUnknown=countListUnknown)

    #set a random allele as reference
    ref(a) <- randomRef(a)
    genotype(a) <- inferGenotypes(a)

    #get the fraction of the reference allele
    refFrac <- fraction(a, top.fraction.criteria="ref")

    #check mean
    mean(refFrac)
    ## [1] 0.499586
```

The reference fraction mean `mean(refFrac)` is in this case very close to 0.5, and suggests that the mapbias globally is low if present. However, in this example we randomly assigned the reference genome to one of the two most expressed alleles by the randomRef function, so it should not have any mapbias.

2 Tests

2.1 Statistical analysis of an ASEset object

One of the simplest statistical test for use in allelic imbalance analysis is the chi-square test. This test assumes that the uncertainty of ASE is represented by a normal distribution around an expected mean (i.e 0.5 for equal expression). A significant result suggests an ASE event. Every SNP, every sample, and every strand is tested independently.

```
    #set ref allele
    ref(a.stranded) <- c("G","T","C")

    #binomial test
    binom.test(a.stranded[,5:8], "+")
    ##               chr17_79478287 chr17_79478331 chr17_79478334
    ## ERR009115.bam            NaN            NaN            NaN
    ## ERR009122.bam   0.7265625000    0.006610751   2.668457e-01
    ## ERR009126.bam   0.0004882813    0.107752144   1.907349e-06
```

```
## ERR009127.bam   0.2500000000    0.500000000   1.000000e+00
#chi-square test
chisq.test(a.stranded[,5:8], "-")
##      chr17_79478287 chr17_79478331 chr17_79478334
## [1,]             NA             NA             NA
## [2,]      0.7962534             NA             NA
## [3,]             NA             NA             NA
## [4,]             NA             NA             NA
```

3 Summary functions

The regionSummary function can be used to investigate if there is a consistent imbalance in the same direction over a region (e.g. a transcript). Besides providing the user with information of how many AI:s are up or down, the function returns several descriptive statistics for the result. The list below explains the acronyms retrieved using the 'basic' accessor:

- hets the number of heterozygote SNPs
- homs the number of homozygote SNPs
- mean.fr the mean of fractions for all hets
- sd.fr the standard deviation for the fractions for all hets
- mean.delta the mean of the delta fractions for all hets
- sd.delta the standard deviation for the delta fractions for all hets
- ai.up the number of hets significantly up-regulated in one allele.
- ai.down the number of hets significantly down-regulated in one allele.

```
# in this example every snp is on separate exons
region <- granges(a.simple)
rs <- regionSummary(a.simple, region)
rs
## class: RegionSummary
## dim: 3 20
## metadata(0):
## assays(1): rs1
## rownames(3): 1 2 3
## rowData names(4): ASEsetIndex regionIndex regionIndexName ASEsetMeta
## colnames(20): individual1 individual2 ... individual19 individual20
## colData names(2): Treatment Gender
basic(rs)
## $`1`
##             hets homs   mean.fr     sd.fr mean.delta   sd.delta ai.up
## individual1    1    0 0.0000000 0.0000000 0.50000000 0.50000000     0
## individual2    1    0 1.0000000 1.0000000 0.50000000 0.50000000     1
## individual3    1    0       NaN       NaN        NaN        NaN     0
## individual4    1    0 0.0000000 0.0000000 0.50000000 0.50000000     0
## individual5    1    0 1.0000000 1.0000000 0.50000000 0.50000000     1
## individual6    1    0 0.4347826 0.4347826 0.06521739 0.06521739     0
## individual7    1    0 0.0000000 0.0000000 0.50000000 0.50000000     0
## individual8    1    0 0.0000000 0.0000000 0.50000000 0.50000000     0
## individual9    1    0       NaN       NaN        NaN        NaN     0
```

```
## individual10    0    1       NaN       NaN        NaN        NaN     0
## individual11    0    1       NaN       NaN        NaN        NaN     0
## individual12    1    0 0.0000000 0.0000000 0.50000000 0.50000000     0
## individual13    1    0 1.0000000 1.0000000 0.50000000 0.50000000     1
## individual14    1    0 0.0000000 0.0000000 0.50000000 0.50000000     0
## individual15    1    0 1.0000000 1.0000000 0.50000000 0.50000000     1
## individual16    1    0 1.0000000 1.0000000 0.50000000 0.50000000     1
## individual17    0    1       NaN       NaN        NaN        NaN     0
## individual18    0    1       NaN       NaN        NaN        NaN     0
## individual19    0    1       NaN       NaN        NaN        NaN     0
## individual20    1    0 1.0000000 1.0000000 0.50000000 0.50000000     1
##              ai.down
## individual1        1
## individual2        0
## individual3        0
## individual4        1
## individual5        0
## individual6        1
## individual7        1
## individual8        1
## individual9        0
## individual10       0
## individual11       0
## individual12       1
## individual13       0
## individual14       1
## individual15       0
## individual16       0
## individual17       0
## individual18       0
## individual19       0
## individual20       0
##
## $`2`
##              hets homs   mean.fr     sd.fr mean.delta  sd.delta ai.up
## individual1     0    1       NaN       NaN        NaN       NaN     0
## individual2     1    0 0.0000000 0.0000000  0.5000000 0.5000000     0
## individual3     0    1       NaN       NaN        NaN       NaN     0
## individual4     0    1       NaN       NaN        NaN       NaN     0
## individual5     0    1       NaN       NaN        NaN       NaN     0
## individual6     0    1       NaN       NaN        NaN       NaN     0
## individual7     0    1       NaN       NaN        NaN       NaN     0
## individual8     1    0 0.9682540 0.9682540  0.4682540 0.4682540     1
## individual9     0    1       NaN       NaN        NaN       NaN     0
## individual10    1    0 0.7105263 0.7105263  0.2105263 0.2105263     1
## individual11    1    0       NaN       NaN        NaN       NaN     0
## individual12    0    1       NaN       NaN        NaN       NaN     0
## individual13    0    1       NaN       NaN        NaN       NaN     0
## individual14    0    1       NaN       NaN        NaN       NaN     0
## individual15    0    1       NaN       NaN        NaN       NaN     0
## individual16    0    1       NaN       NaN        NaN       NaN     0
```

```
## individual17    0    1       NaN       NaN        NaN       NaN     0
## individual18    0    1       NaN       NaN        NaN       NaN     0
## individual19    0    1       NaN       NaN        NaN       NaN     0
## individual20    1    0 0.7656250 0.7656250  0.2656250 0.2656250     1
##              ai.down
## individual1        0
## individual2        1
## individual3        0
## individual4        0
## individual5        0
## individual6        0
## individual7        0
## individual8        0
## individual9        0
## individual10       0
## individual11       0
## individual12       0
## individual13       0
## individual14       0
## individual15       0
## individual16       0
## individual17       0
## individual18       0
## individual19       0
## individual20       0
##
## $`3`
##              hets homs   mean.fr     sd.fr mean.delta  sd.delta ai.up
## individual1     1    0 0.8076923 0.8076923  0.3076923 0.3076923     1
## individual2     1    0 1.0000000 1.0000000  0.5000000 0.5000000     1
## individual3     1    0 1.0000000 1.0000000  0.5000000 0.5000000     1
## individual4     1    0 0.0000000 0.0000000  0.5000000 0.5000000     0
## individual5     1    0 1.0000000 1.0000000  0.5000000 0.5000000     1
## individual6     1    0 0.8615385 0.8615385  0.3615385 0.3615385     1
## individual7     1    0 0.6428571 0.6428571  0.1428571 0.1428571     1
## individual8     0    1       NaN       NaN        NaN       NaN     0
## individual9     1    0       NaN       NaN        NaN       NaN     0
## individual10    0    1       NaN       NaN        NaN       NaN     0
## individual11    1    0       NaN       NaN        NaN       NaN     0
## individual12    0    1       NaN       NaN        NaN       NaN     0
## individual13    0    1       NaN       NaN        NaN       NaN     0
## individual14    1    0 1.0000000 1.0000000  0.5000000 0.5000000     1
## individual15    0    1       NaN       NaN        NaN       NaN     0
## individual16    0    1       NaN       NaN        NaN       NaN     0
## individual17    1    0       NaN       NaN        NaN       NaN     0
## individual18    0    1       NaN       NaN        NaN       NaN     0
## individual19    1    0 0.0000000 0.0000000  0.5000000 0.5000000     0
## individual20    1    0 0.1836735 0.1836735  0.3163265 0.3163265     0
##              ai.down
## individual1        0
## individual2        0
```

```
## individual3        0
## individual4        1
## individual5        0
## individual6        0
## individual7        0
## individual8        0
## individual9        0
## individual10       0
## individual11       0
## individual12       0
## individual13       0
## individual14       0
## individual15       0
## individual16       0
## individual17       0
## individual18       0
## individual19       1
## individual20       1
```

4 Base graphics

4.1 Plotting of an ASEset object

The barplot function for ASEset objects plots the read count of each allele in each sample. This is useful for getting a detailed view of individual SNPs in few samples.

```
barplot(a.stranded[2], strand="+", xlab.text="", legend.interspace=2)
```

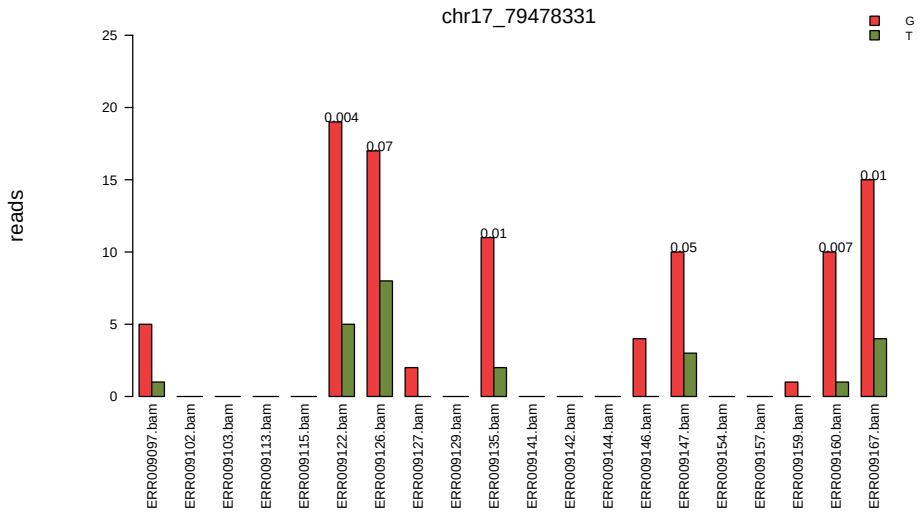

Figure 1: The red bars show how many reads with the G-allele that overlaps the snp at position chr17:79478331, and the green bars show how many reads with the T allele that overlaps

As can be seen in figure 1 several samples from the HapMap data show a strong imbalance at the chr17:79478331 position on the plus strand. By default the p-value is calculated by a chi-square test, and when the counts for one allele are below 5 for one allele the chi-square test returns NA, which is why there is no P-value above the first bar in the figure 1. The default p-values can be substitued by other test results via the arguments testValue and testValue2.

```
#use another source of p-values
btp <- binom.test(a.stranded[1],"+")
barplot(a.stranded[2], strand="+", testValue=t(btp), xlab.text="",
            legend.interspace=2)
```

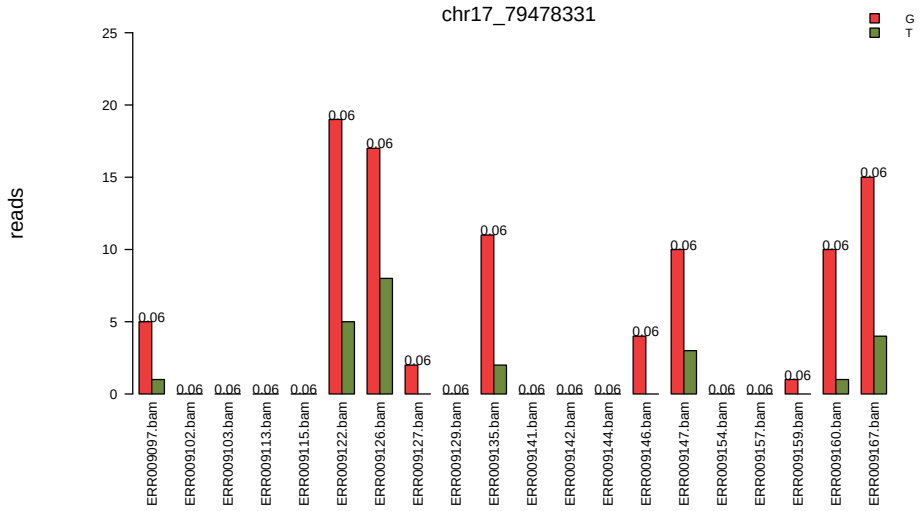

Figure 2: Here the default chi-square p-values have been replaced by p-values from binomial tests

In the figure 2 the binomial test has been used instead of the default chi-square test. At low counts the P-value can differ, but as can be seen in the table below the difference matters less with more read counts.

```
init <- c(15,20)
for(i in c(1,2,4,6)){
    bp <- signif(binom.test(init*i, p=0.5)$p.value,3)
    cp <- signif(chisq.test(init*i, p=c(0.5,0.5))$p.value, 3)
    cat(paste("A: ", init[1]*i , " B: ", init[2]*i, " binom p: ", bp, "chisq p: ", cp,"\n"))
}
## A:  15  B:  20  binom p:  0.5 chisq p:  0.398
## A:  30  B:  40  binom p:  0.282 chisq p:  0.232
## A:  60  B:  80  binom p:  0.108 chisq p:  0.091
## A:  90  B:  120  binom p:  0.0451 chisq p:  0.0384
```

Another type of barplot can be invoked by the argument type="fraction". This plotting mechanism is useful to illustrate more SNPs and more samples, using less space than the standard barplot with the default type="counts".

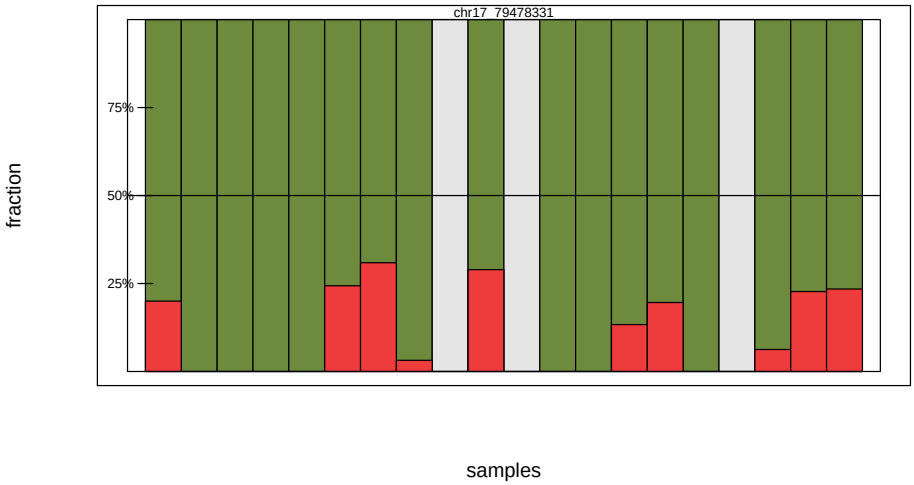

Figure 3: A barplot with type='fraction'
Each bar represents one sample and by default the most expressed allele over all samples is on the top, here in green. The black line denotes 1:1 expression of the alleles.

```
barplot(a.simple[2], type="fraction", cex.main = 0.7)
```

A typical question would be to ask why certain heterozygote samples have allele specific expression. The arguments sampleColour.top and sampleColour.bot allows for different highligts such as illustrated here below for gender. This could also be used to highlight based on genotype of proximal non-coding SNPs if available.

```
#top and bottom colour
sampleColour.top <-rep("palevioletred",ncol(a.simple))
sampleColour.top[colData(a.simple)[,"Gender"]%in%"male"] <- "darkgreen"
sampleColour.bot <- rep("blue",ncol(a.simple))
sampleColour.bot[colData(a.simple)[,"Gender"]%in%"male"] <- "seashell2"
barplot(a.simple[2], type="fraction", sampleColour.top=sampleColour.top,
    sampleColour.bot=sampleColour.bot, cex.main = 0.7)
```

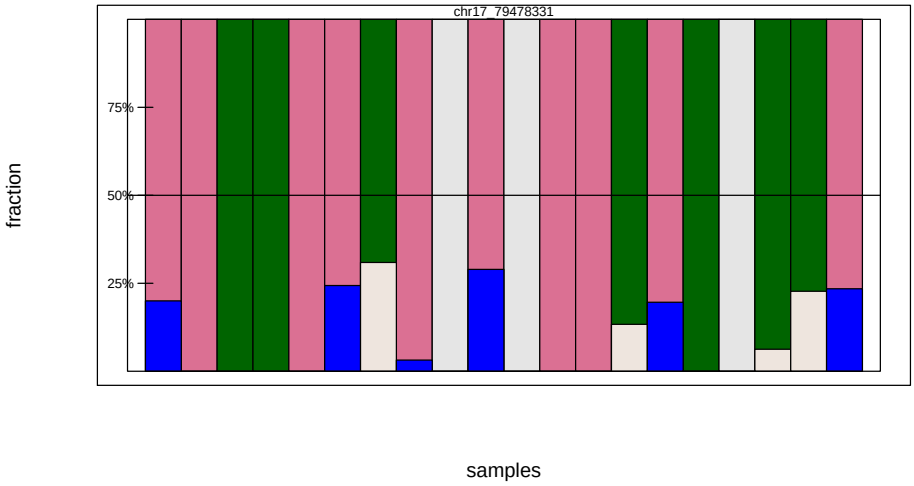

Figure 4: A barplot with allele fractions, additionally colored by gender

4.2 Plot with annotation

It is often of interest to combine the RNA sequencing data with genomic annotation information from online databases. For this purpose there is a function to extract variant specific annotation such as gene, exon, transcript and CDS.

```
library(org.Hs.eg.db)
barplot(a.simple[1],OrgDb=org.Hs.eg.db,
            cex.main = 0.7,
            xlab.text="",
            ypos.annotation = 0.08,
            annotation.interspace=1.3,
            legend.interspace=1.5
            )
```

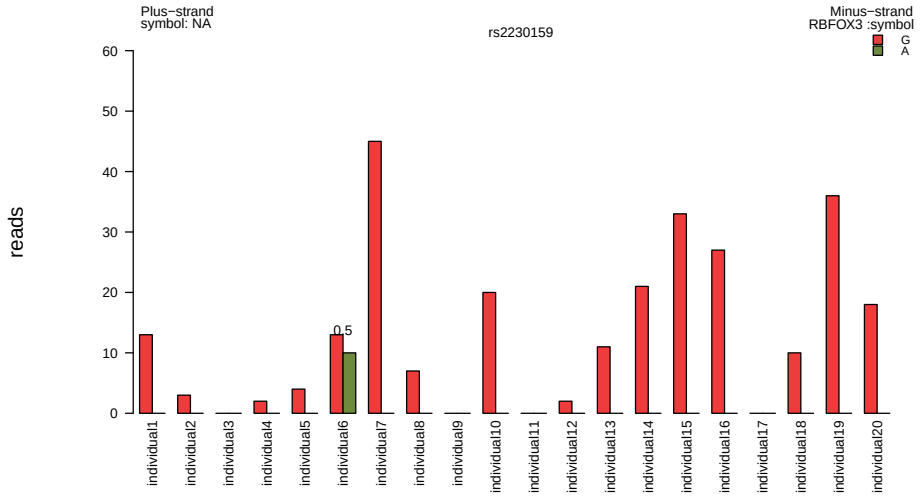

Figure 5: A barplot with Gene information extracted from the org.Hs.eg.db package

##Top allele criteria The barplot with type='fraction' can be visualized according to three different allele sorting criteria. The default behaviour is just to shown the allele with highest overall abundance in the upper half of the plot. This works well for most single SNP investigations. For more complex situations, however, it can be essential to keep track of phase information. This is done either through the reference allele sorting function, or even better, through consistently showing the maternal allele on top. When phase is know, this is essential to compare effect-directions of different coding SNPs

```
#load data
data(ASEset)

#using reference and alternative allele information
alt(ASEset) <- inferAltAllele(ASEset)
barplot(ASEset[1], type="fraction", strand="+", xlab.text="",
            top.fraction.criteria="ref", cex.main=0.7)
```

```
#using phase information
phase(ASEset) <- phase.mat
```

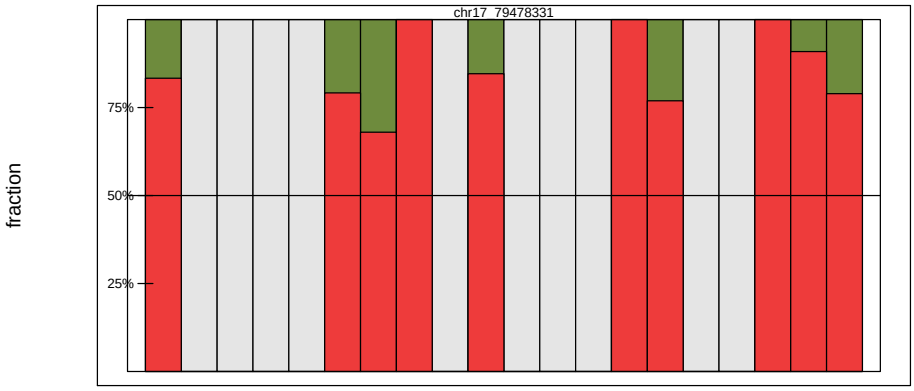

Figure 6: A barplot with the reference allele as top fraction, top.fraction.criteria='ref'

```
barplot(ASEset[1], type="fraction", strand="+", xlab.text="",
            top.fraction.criteria="phase", cex.main=0.7)
```

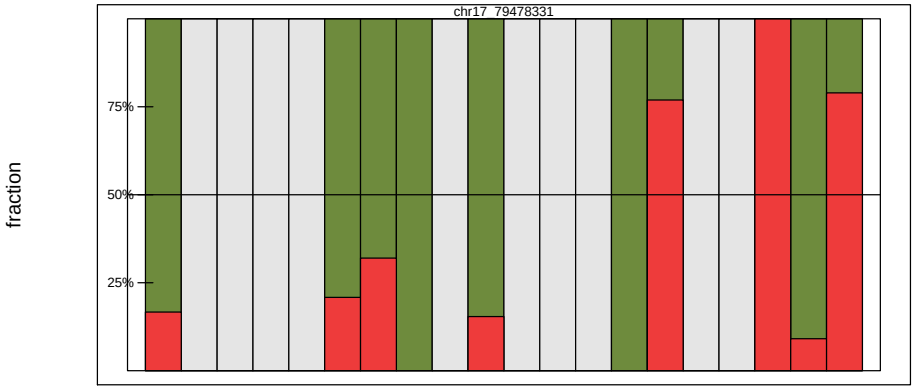

Figure 7: A barplot with the maternal phase as top fraction, top.fraction.criteria='phase'

4.3 locationplot

Finally a given gene or set of proximal genes will often have several SNPs close to each other. It is of interest to investigate all of them together, in connection with annotation information. This can be done using the locationplot function. This function in its simplest form just plot all the SNPs in an ASEset distributed by genomic location. Additionally it contains methods for including gene-map information through the arguments OrgDb and TxDb.

The overview offers the possibility to view consistency of SNP fractions over consecutive exons which is important to assess the reliability of the measured SNPs. In the best case all fractions deviates with the same magnitude for every SNP in the same gene for an individual. To be able to inspect that the direction is right, it is possible to use the top.fraction.criteria and use phase information as explained earlier.

```
# locationplot using count type
a.stranded.sorted <- sort(a.stranded, decreasing=FALSE)
locationplot(a.stranded.sorted, type="count", cex.main=0.7, cex.legend=0.4)
```

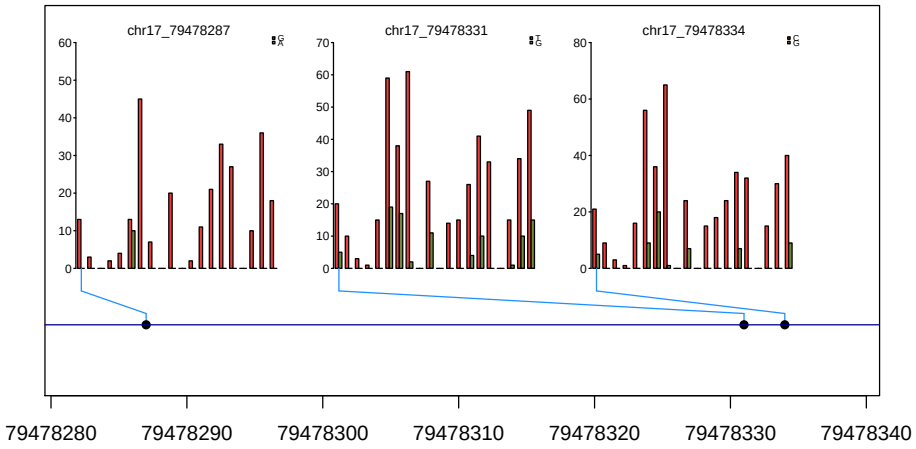

Figure 8: A locationplot with countbars displaying a region of SNPs

```
# locationplot annotation
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
locationplot(a.stranded.sorted, OrgDb=org.Hs.eg.db,
             TxDb=TxDb.Hsapiens.UCSC.hg19.knownGene)
```

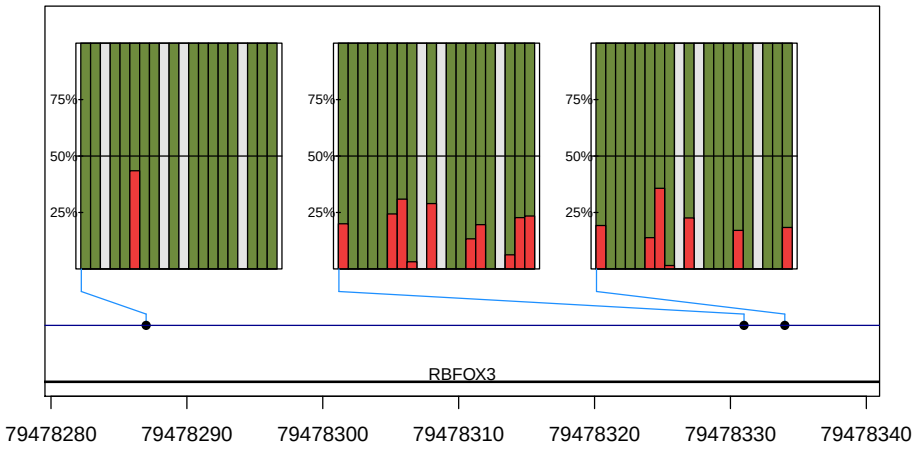

Figure 9: A locationplot with Gene and Transcript information extracted from the org.Hs.eg.db and TxDb.Hsapiens.UCSC.hg19.knownGene packages

5 Grid graphics

To make use of the extra graphical flexibility that comes with grid graphics, the AllelicImbalance package now has functions that can be integrated in the grid environment. The low level grid functions are located in the *grid package*, but higher level functions based on grid can be

found in popular packages such as e.g. *lattice*, *ggplot2*. The benefits using grid graphics is a better control over different graphical regions and coordinate systems, how to construct and redirect graphical output, such as lines, points, etc and their appearance.

One particular reason for the AllelicImbalance to use grid graphics is the *Gviz* package, which uses grids functionality to construct tracks to visualize genomic regions in a smart way. The AllelicImbalance package has grid based barplots and functionality to create Gviz tracks. Observe the extra "g" prefix for the bar- and locationplot versions using the grid, see examples below.

5.1 gbarplot

The standard barplot functions for ASEset objects have at the moment much more parameters that can be set by the user than the gbarplot. Base graphics also produce graphs faster than the grid environment. The advantage of the gbarplot is instead the possiblity to integrate into the Gviz package and other grid based environments.

```
#gbarplots with type "count"
gbarplot(ASEset, type="count")
```

```
# gbarplots with type "fraction"
gbarplot(ASEset, type="fraction")
```

5.2 glocationplot

The glocationplot is a wrapper to quickly get an overview of ASE in a region, similar to locationplot(), but built with the grid graphics track system from the Gviz package.

```
#remember to set the genome
genome(ASEset) <- "hg19"

glocationplot(ASEset, strand='+')
```

5.3 Custom location plots

More flexibility and functionality from the Gviz package is accessed if the tracks are constructed separately. This is useful to evalute AI in respect to for example read coverage or transcript annotation. An inconsistency of AI could potentially be explained by an uneven coverage over the exons, or lack of coverage. Below is an example using two samples from the ASEset and the corresponding reads. Ensuring and setting the seqlevels equal is a security measure, for the internal overlap calculations that otherwise might throw a warning. The GR object is defining the region that we want to plot. The ASEDAnnotationTrack will create a track based on the gbarplots and the CoverageDataTrack a track based on read coverage. The different tracks are then collected in a list with the first element appearing at the top in the final plotting by the plotTracks function. The sizes vector defines the vertical space assigned to each track. To use transcript annotation, more detailed information can be found in the Gviz vignette

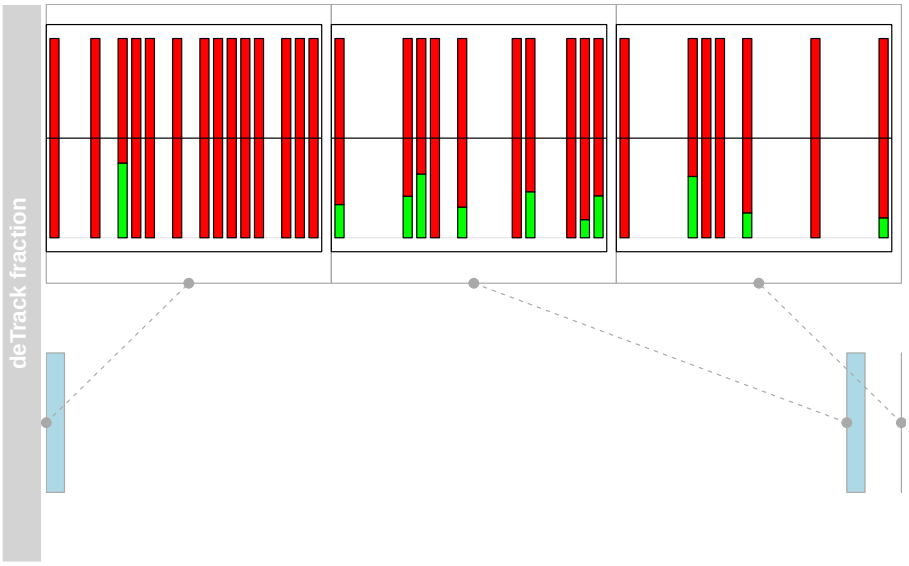

Figure 10: A glocationplot, showing the ASE of SNPs within a defined region

```
#load Gviz package
library(Gviz)

#subset ASEset and reads
x <- ASEset[,1:2]
r <- reads[1:2]
seqlevels(r, pruning.mode="coarse") <- seqlevels(x)

GR <- GRanges(seqnames=seqlevels(x),
        ranges=IRanges(start=min(start(x)), end=max(end(x))),
        strand='+', genome=genome(x))

deTrack <- ASEDAnnotationTrack(x, GR=GR, type='fraction', strand='+')
covTracks <- CoverageDataTrack(x, BamList=r, strand='+')
lst <- c(deTrack,covTracks)
sizes <- c(0.5,rep(0.5/length(covTracks),length(covTracks)))

plotTracks(lst, from=min(start(x)), to=max(end(x)), sizes=sizes,
                col.line = NULL, showId = FALSE, main = 'main', cex.main = 1,
                title.width = 1, type = 'histogram')
```

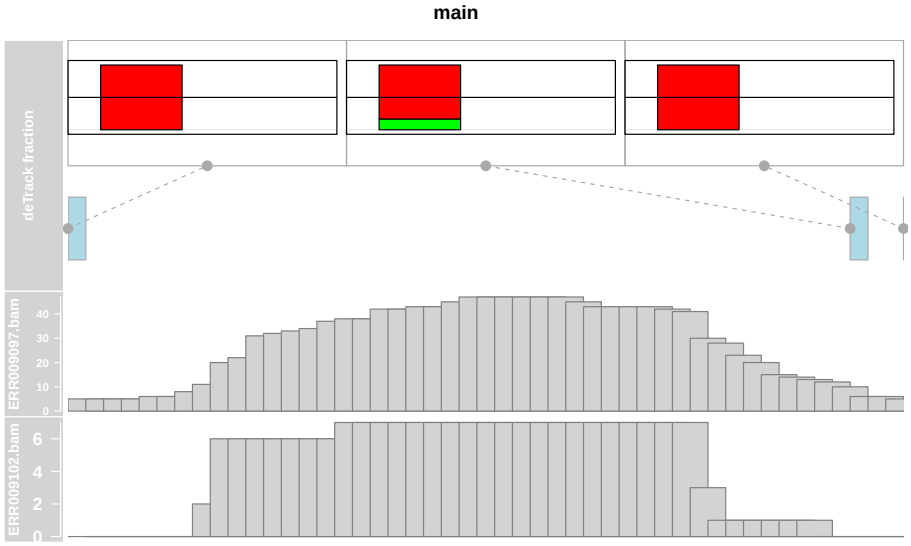

Figure 11: A custom built Gviz plot using the ASEDAnnotationTrack, showing the ASE of SNPs within a defined region together with read coverage information

6 Important notifications

6.1 Update old objects

Due to changes in the RangedSummarizedExperiment class from the *SummarizedExperiment* package, all RangedSummarizedExperiment objects prior the release of Bioconductor 3.2 needs to be updated. The ASEset is based on the RangedSummarizedExperiment class and solely needs to be updated as well.

```
data(ASEset.old)
## Warning in data(ASEset.old): data set 'ASEset.old' not found
#this command doesnt work anymore, for some reason(Will ask the mail-list)
#ASEset.new <- updateObject(ASEset.old)
```

7 Conclusion

In conclusion we hope that you will find this package useful in the investigation of the genetics of RNA-seq experiments. The various import functions should assist in the task of actually retrieving allele counts for specific nucleotide positions from all RNA-seq reads, including the non-trivial cases of intron-spanning reads. Likewise, the statistical analysis and plotting functions should be helpful in discovering any allele specific expression patterns that might be found in your data.

8 Links

Bowtie link

BWA link

Samtools link

Samtools pileup link

Grid graphics link

9 Session Info

```
sessionInfo()
## R version 3.5.0 (2018-04-23)
## 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/3.5/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRlapack.dylib
##
## locale:
## [1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
##  [1] parallel  stats4    grid      stats     graphics  grDevices utils
##  [8] datasets  methods   base
##
## other attached packages:
##  [1] Gviz_1.24.0
##  [2] TxDb.Hsapiens.UCSC.hg19.knownGene_3.2.2
##  [3] GenomicFeatures_1.32.0
##  [4] SNPlocs.Hsapiens.dbSNP144.GRCh37_0.99.20
##  [5] BSgenome_1.48.0
##  [6] rtracklayer_1.40.0
##  [7] org.Hs.eg.db_3.6.0
##  [8] AnnotationDbi_1.42.0
##  [9] VariantAnnotation_1.26.0
## [10] AllelicImbalance_1.18.0
## [11] GenomicAlignments_1.16.0
## [12] Rsamtools_1.32.0
## [13] Biostrings_2.48.0
## [14] XVector_0.20.0
## [15] SummarizedExperiment_1.10.0
## [16] DelayedArray_0.6.0
## [17] BiocParallel_1.14.0
## [18] matrixStats_0.53.1
## [19] Biobase_2.40.0
```

```
## [20] GenomicRanges_1.32.0
## [21] GenomeInfoDb_1.16.0
## [22] IRanges_2.14.0
## [23] S4Vectors_0.18.0
## [24] BiocGenerics_0.26.0
## [25] BiocStyle_2.8.0
##
## loaded via a namespace (and not attached):
##  [1] nlme_3.1-137           ProtGenerics_1.12.0    bitops_1.0-6
##  [4] bit64_0.9-7            RColorBrewer_1.1-2     progress_1.1.2
##  [7] httr_1.3.1             rprojroot_1.3-2        tools_3.5.0
## [10] backports_1.1.2        R6_2.2.2               rpart_4.1-13
## [13] Hmisc_4.1-1            DBI_0.8                lazyeval_0.2.1
## [16] colorspace_1.3-2       ade4_1.7-11            nnet_7.3-12
## [19] gridExtra_2.3          prettyunits_1.0.2      bit_1.1-12
## [22] curl_3.2               compiler_3.5.0         htmlTable_1.11.2
## [25] bookdown_0.7           scales_0.5.0           checkmate_1.8.5
## [28] stringr_1.3.0          digest_0.6.15          foreign_0.8-70
## [31] rmarkdown_1.9          pkgconfig_2.0.1        base64enc_0.1-3
## [34] dichromat_2.0-0        htmltools_0.3.6        ensembldb_2.4.0
## [37] htmlwidgets_1.2        rlang_0.2.0            rstudioapi_0.7
## [40] RSQLite_2.1.0          acepack_1.4.1          RCurl_1.95-4.10
## [43] magrittr_1.5           GenomeInfoDbData_1.1.0 Formula_1.2-2
## [46] Matrix_1.2-14          Rcpp_0.12.16           munsell_0.4.3
## [49] stringi_1.1.7          yaml_2.1.18            MASS_7.3-50
## [52] zlibbioc_1.26.0        plyr_1.8.4             blob_1.1.1
## [55] lattice_0.20-35        splines_3.5.0          knitr_1.20
## [58] pillar_1.2.2           seqinr_3.4-5           biomaRt_2.36.0
## [61] XML_3.98-1.11          evaluate_0.10.1        biovizBase_1.28.0
## [64] latticeExtra_0.6-28    data.table_1.10.4-3    gtable_0.2.0
## [67] assertthat_0.2.0       ggplot2_2.2.1          xfun_0.1
## [70] AnnotationFilter_1.4.0 survival_2.42-3        tibble_1.4.2
## [73] memoise_1.1.0          cluster_2.0.7-1
```