RiboProfiling - Analysis and quality assessment of Ribosome-profiling data

Alexandra Popa

May 19, 2021

Contents

1 Introduction

Ribosome profiling, the recently developed high throughput sequencing technique, enabled the mapping of translated regions genome-wide. This technique takes advantage of the fact that ribosomes actively engaged in translation can protect their associated mRNA fragments against RNAse digestion. The sequencing of these protected fragments can reveal potentially translated sequences. *RiboProfiling* is a Bioconductor package that provides multiple types of ribo-seq data analysis, starting from BAM files alone:

- Quality assessment of read match size distribution.
- Read coverage around the TSS for the definition of an offset (shift value) of ribosome position.
- Calibrate reads by applying an offset to the read start positions along the transcript.
- Table of read start counts (shifted if specified) on the specified region (CDS, 5pUTR, 3pUTR).
- A graphical quality function for ribo-seq data, based on the previously obtained tables.
- The quantification of the frequency and coverage matrices for motifs of codons (1, 2 or 3 codons) in ORFs.
- Principal component analysis of codon motif coverage in ribosome-profiling.

The package also provides the `riboSeqFromBAM` function, that assembles in a global framework some of the above analyses, allowing a quick overview of the data.

2 Data Input

As input data, the *RiboProfiling* package can start with a path to a ribo-seq (or other type of) BAM to be analyzed. A list of multiple BAM files can also be provided. BAM files from either bowtie/tophat, STAR, and Lifescope (Solid), single- and paired-end have been validated. Many intermediate functions of the *RiboProfiling* package can be called for additional modifications on the data objects.

Example data files provided with the *RiboProfiling* package are based on chromosome 1 reads from human primary BJ fibroblasts data (PMID: 23594524, SRA: http://www.ebi.ac.uk/ena/data/view/SRP020544):

- BAM file Ctrl "http://genomique.info/data/public/RiboProfiling/ctrl.bam".
- BAM file Nutlin2h "http://genomique.info/data/public/RiboProfiling/nutlin2h.bam".

The data provided with the *RiboProfiling* package and accessible through the `data` function consists of:

- **ctrlGAlignments** - a GAlignments object corresponding to the "http://genomique.info/data/public/RiboProfiling/ctrlI.bam" BAM.
- **codonIndexCovCtrl** - a list containing the number of reads for each codon in each ORF.
- **codonDataCtrl** - a list of 2 data.frames containing the frequency of codons for each ORF and, respectively, the read coverage for the same codons and ORFs.

3 Quick start

The fastest way to analyze a list of BAM files with the *RiboProfiling* package is to call the `riboSeqFromBAM` function. The only input needed are the paths to the BAM files and the genome version on which the mapping was done. Here is an example on how to use this wrapper function on 2 BAM files.

```
library(RiboProfiling)
listInputBam <- c(
    BamFile("http://genomique.info/data/public/RiboProfiling/ctrl.bam"),
    BamFile("http://genomique.info/data/public/RiboProfiling/nutlin2h.bam")
    )
covData <- riboSeqFromBAM(listInputBam, genomeName="hg19")
```

The following analyses and results are returned:

- **a histogram of read match lengths**: figures 1 and 2.
- **a read start coverage plot around the TSS**: figures 1 and 2.
- **an automatic estimation and application of the ribosome offset position (if no offset value is provided (listShiftValue parameter missing))**
- **a data.frame containing CDSs annotation and counts on the 5pUTR, CDS, and 3pUTR once the offset has been applied**
- **pairs and boxplots of the read counts**: figures 3 and 4.
- **a list of per ORF per codon coverage**

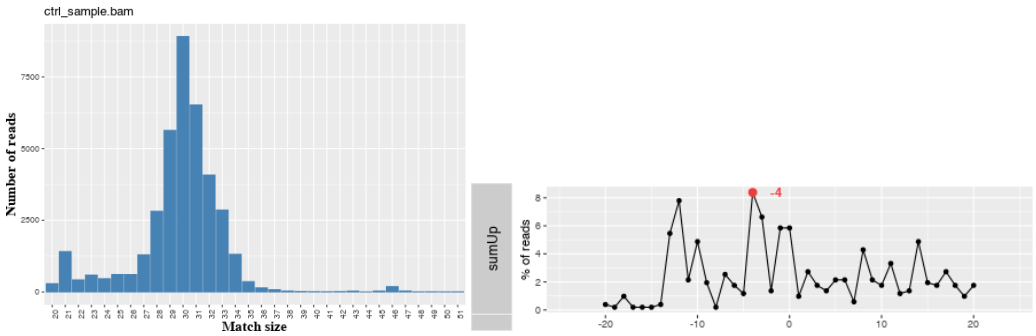

Figure 1: Histogram of read match size and read start offset to ribosome P-site for ctrl.bam dataset.

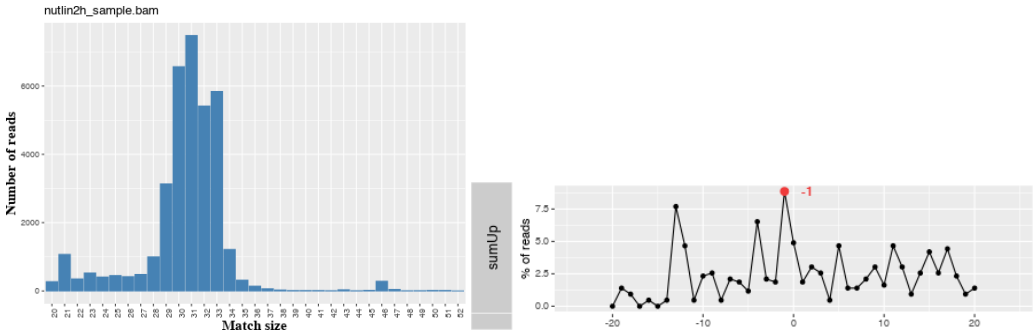

Figure 2: Histogram of read match size and read start offset to ribosome P-site for nutlin2h.bam dataset.

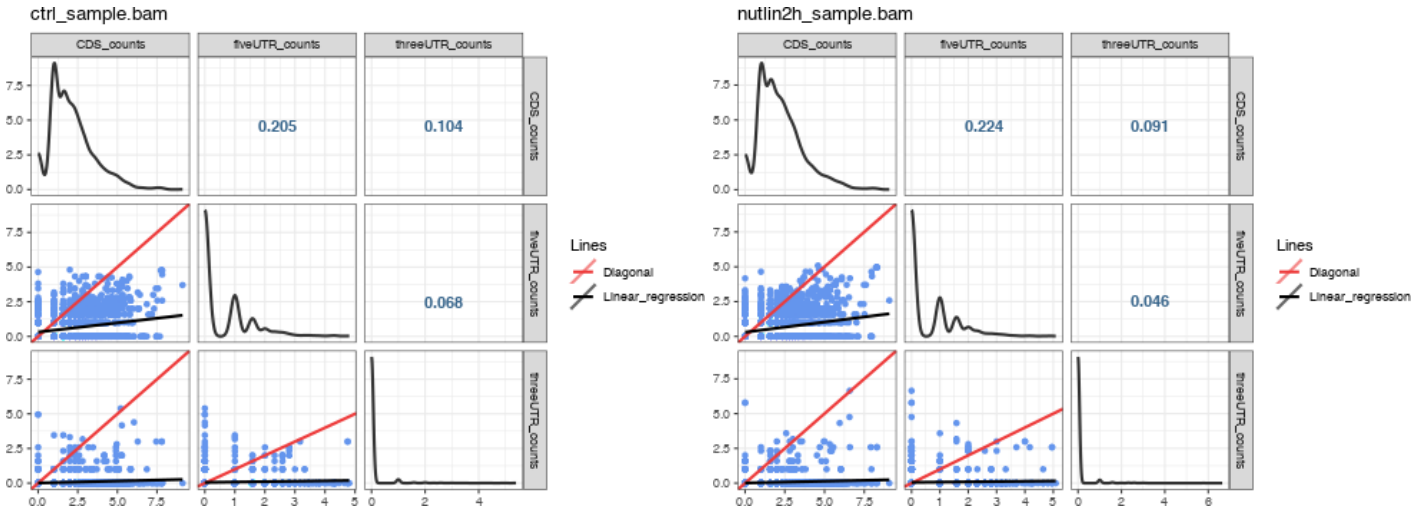

Figure 3: Pairs of shifted read counts on 5pUTR, CDS, and 3pUTR for ctrl.bam and nutlin2h.bam dataset.

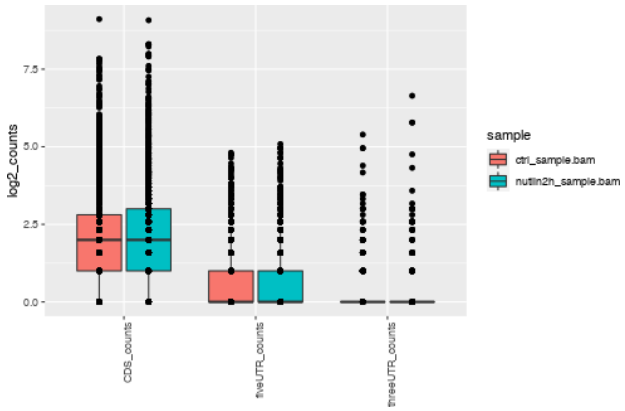

Figure 4: **Boxplot of shifted read counts on 5pUTR, CDS, and 3pUTR for ctrl.bam and nutlin2h.bam dataset.**

All these results will be explained in detail in the following sections.

This general function only works if your species model has an ensembl table in the UCSC database or if you provide the corresponding txdb object containing the annotations.

4 Histogram of read match length

This histogram represents both a quality control of the sequencing and an important tool to define the match sizes of reads corresponding to ribosome footprints (around 30bp). The *histMatchLength* function in the *RiboProfiling* package produces this histogram starting from a GAlignment object (the loaded records of a BAM) such as the *ctrlGAlignments* data example (figure 5).

One can create a GAlignments from a BAM using the `readGAlignments` function from the *GenomicAlignments* package:

```
aln <- readGAlignments(
        BamFile("http://genomique.info/data/public/RiboProfiling/ctrl.bam")
    )
```

Or based on an already existing GAlignments object:

```
data(ctrlGAlignments)
aln <- ctrlGAlignments
matchLenDistr <- histMatchLength(aln, 0)
matchLenDistr[[2]]
```

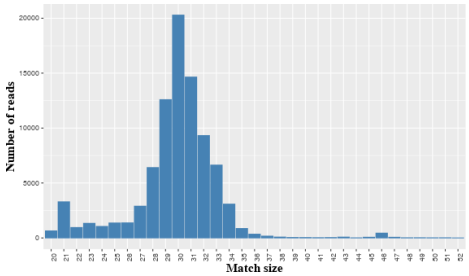

Figure 5: **Histogram of read match length for example data: ctrlGAlignments.**

5 Read start coverage plot around the TSS

The initiation of protein synthesis starts with the first codon in the P-site. In ribosome profiling experiments, the location of the P-site codon must be inferred in order to recalibrate the read start positions relative to the transcript. An offset is usually observed in Ribo-seq experiments between the starting of the reads and the AUG codons of protein coding sequences. Three functions allow the estimation of the read start position relative to the P-site codon:

- `aroundPromoter`: returns the genomic positions flanking the transcript start site (TSS) for the x% (3% default value) best expressed CDSs.
- `readStartCov`: returns the read start coverage around the TSS on the predefined CDSs.
- `plotSummarizedCov`: plots the summarized coverage in a specified range (e.g. around TSS) for the specified match sizes.

```
#transform the GAlignments object into a GRanges object
#(faster processing of the object)
alnGRanges <- readsToStartOrEnd(aln, what="start")
#txdb object with annotations
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
oneBinRanges <- aroundPromoter(txdb, alnGRanges, percBestExpressed=0.001)
#the coverage in the TSS flanking region for the reads with match sizes 29:31
listPromoterCov <-
    readStartCov(
        alnGRanges,
        oneBinRanges,
        matchSize=c(29:31),
        fixedInterval=c(-20, 20),
        renameChr="aroundTSS",
        charPerc="perc"
    )
plotSummarizedCov(listPromoterCov)
```

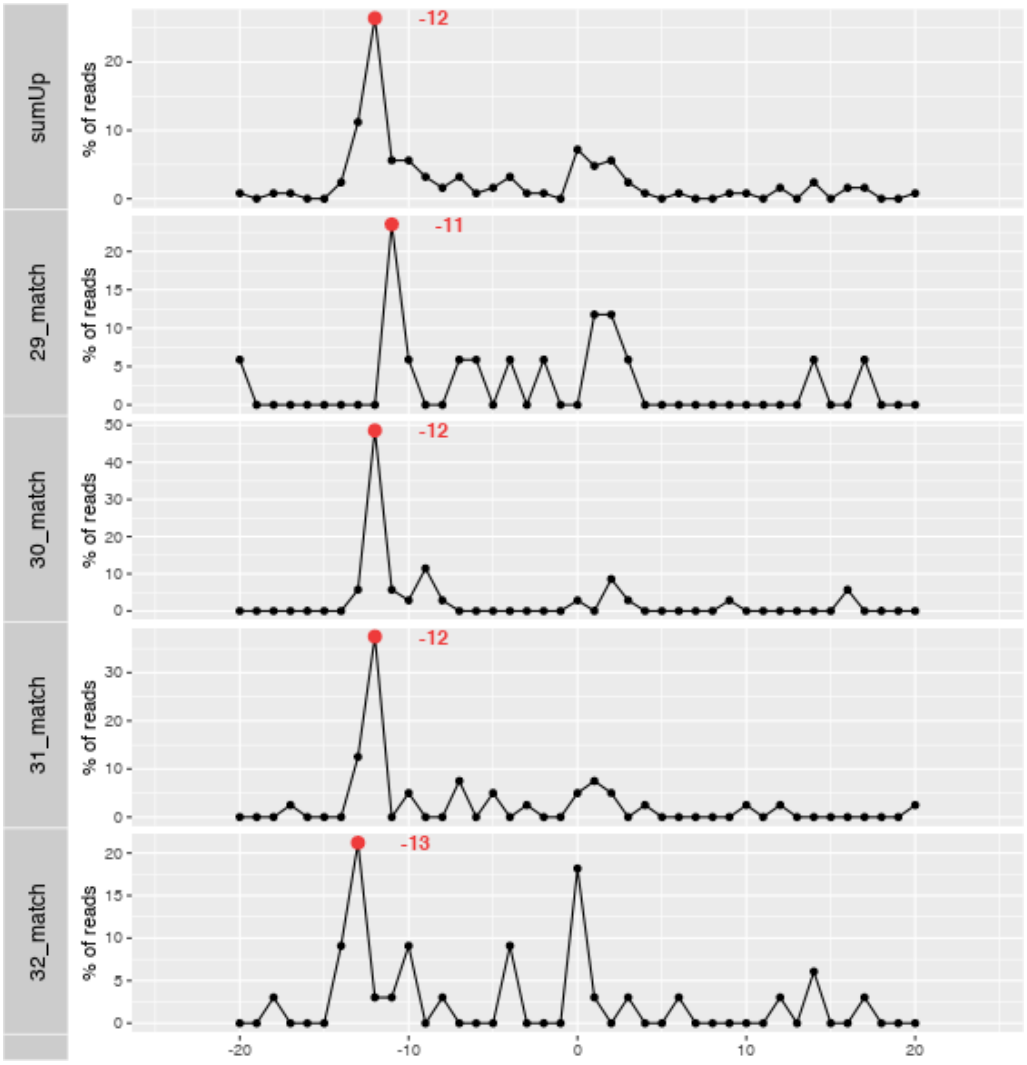

Figure 6: **Offset around TSS for the best expressed CDSs on a subset of match sizes.**

The first time you analyse a ribo-seq BAM file, it is advisable to make an offset graph for all the previously selected read match sizes, as the offset might be different depending on the read match size .

6 Count reads on features

The purpose of Ribosome Profiling experiments is mainly to identify RNAse resistant regions, which might indicate that these sequences are likely to be translated. The `countShiftReads` function quantifies the read start coverage on different sequence features. This function integrates the specificity of ribo-seq data of having shifted reads starts from the P-site codon. The `countShiftReads` function recalibrates the read start positions along the transcript with the specified offset (parameter *shiftValue*, default 0) and returns the coverage on the 5pUTR, CDS, 3pUTR, and a matrix of codon coverage for each ORF.

The counts can be used globally to see what is RNAse protected and what is not, but also for differential analysis between conditions. The `countsPlot` function provides some quality graphs for the ribo-seq data: pairs between counts on features and boxplots of counts between conditions.

As an example, we compute the read coverage on the **ctrlGAlignments** example data:

```
#keep only the match read sizes 30-33
alnGRanges <- alnGRanges[which(!is.na(match(alnGRanges$score,30:33)))]
#get all CDSs by transcript
cds <- cdsBy(txdb, by="tx", use.names=TRUE)
#get all exons by transcript
exonGRanges <- exonsBy(txdb, by="tx", use.names=TRUE)
#get the per transcript relative position of start and end codons
cdsPosTransc <- orfRelativePos(cds, exonGRanges)
#compute the counts on the different features
#after applying the specified shift value on the read start along the transcript
countsDataCtrl1 <-
    countShiftReads(
        exonGRanges=exonGRanges[names(cdsPosTransc)],
        cdsPosTransc=cdsPosTransc,
        alnGRanges=alnGRanges,
        shiftValue=-14
    )
head(countsDataCtrl1[[1]])
listCountsPlots <- countsPlot(
    list(countsDataCtrl1[[1]]),
    grep("_counts$", colnames(countsDataCtrl1[[1]])),
    1
)
listCountsPlots
```

```
## Warning in countShiftReads(exonGRanges[names(cdsPosTransc)], cdsPosTransc, :
Param motifSize should be an integer!  Accepted values 3, 6 or 9.  Default value
is 3.

## Warning:  Ignoring unknown aesthetics:  fill

##                 gene  chr strand transc_genomic_start transc_genomic_end
## uc001abw.1 uc001abw.1 chr1      +               861121             879961
## uc031pjl.1 uc031pjl.1 chr1      +               861302             879533
## uc031pjm.1 uc031pjm.1 chr1      +               861302             879533
##            transc_length orf_start orf_end orf_length CDS_counts fiveUTR_counts
## uc001abw.1          2554        81    2126       2046          2              0
## uc031pjl.1          2120        21    2120       2100          2              0
## uc031pjm.1          2084        21    2084       2064          2              0
##            threeUTR_counts
## uc001abw.1               0
## uc031pjl.1               0
## uc031pjm.1               0
```

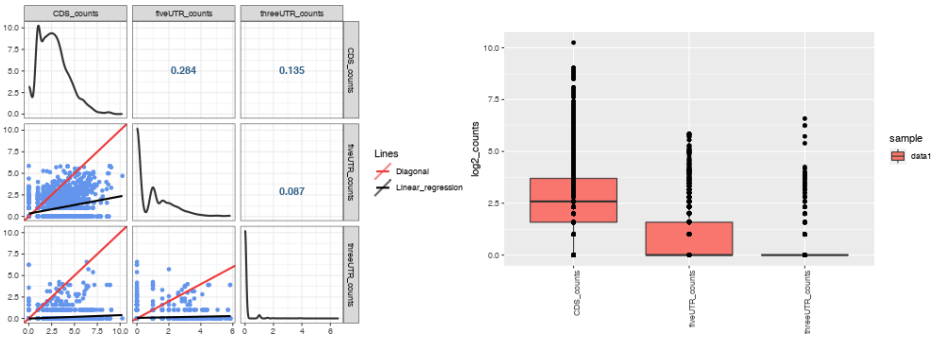

Figure 7: Pairs and boxplots of read coverage on genomic features.

7 Count reads on codon motifs

The previous `countShiftReads` function produces a second element in the list: the counts of reads per motifs of 1, 2, or 3 codons. For each ORF, for each motif in the ORF, the Ribo-seq coverage of the motif is reported as follows:

- for motifs of 3 nucleotides (1 codon) - the sum of read starts on the 3 nucleotides in the codon is reported.
- consecutive motifs of 6 nucleotides (2 consecutive codons) overlap on 3 nucleotides. The Ribo-seq coverage is reported as the coverage on the 1st codon in the motif considered as being in the P-site.
- consecutive motifs of 9 nucleotides (3 consecutive codons) overlap on 6 nucleotides. The Ribo-seq coverage is reported as the coverage on the 2nd codon in the motif considered as being in the P-site.

Codon motifs in each ORF are described as the index of their position in the ORF.

Here is an example:

```
data(codonIndexCovCtrl)
head(codonIndexCovCtrl[[1]], n=3)
```

```
##   codonID nbrReads
## 1       1        0
## 2       2        0
## 3       3        0
```

The `codonInfo` function associates the read coverage on codons with their corresponding codon type for each ORF. It returns a list of 2 matrices: one containing the frequency of each codon motif type in each ORF and the second containing the coverage of each codon motif type in each ORF.

```
listReadsCodon <- countsDataCtrl1[[2]]
#get the names of the expressed ORFs grouped by transcript
cds <- cdsBy(txdb, use.names=TRUE)
orfCoord <- cds[names(cds) %in% names(listReadsCodon)]
#chromosome names should correspond between the BAM,
#the annotations, and the genome
genomeSeq <- BSgenome.Hsapiens.UCSC.hg19
#codon frequency, coverage, and annotation
```

```
codonData <- codonInfo(listReadsCodon, genomeSeq, orfCoord)
```

```
## Warning in codonInfo(listReadsCodon, genomeSeq, orfCoord): Param motifSize should
be an integer! Accepted values 3, 6 or 9. Default value is 3.

## Warning in result_fetch(res@ptr, n = n): SQL statements must be issued with
dbExecute() or dbSendStatement() instead of dbGetQuery() or dbSendQuery().
```

```
data("codonDataCtrl")
head(codonDataCtrl[[1]], n=3)
```

```
##            AAA AAC AAG AAT ACA ACC ACG ACT AGA AGC AGG AGT ATA ATC ATG ATT CAA
## uc010nxq.1   1   2   1   0   1   2   2   2   2   3   6   6   0   3   3   0   0
## uc001abv.1   0   4   6   0   1   3   0   1   3   8   1   1   2   5   3   1   0
## uc001abw.1   1   8  18   2   4  13   7   4   3  21  11   2   2   8   8   1   4
##            CAC CAG CAT CCA CCC CCG CCT CGA CGC CGG CGT CTA CTC CTG CTT GAA GAC
## uc010nxq.1   7   5   2   3   9   0   4   0   0   0   0   1   3   8   2   1   3
## uc001abv.1   5   5   2   1   5   4   2   4   3   5   1   0   2   6   1   1   6
## uc001abw.1  16  27   6  14  45  13  12   8  10  21   3   1  13  56   5   5  22
##            GAG GAT GCA GCC GCG GCT GGA GGC GGG GGT GTA GTC GTG GTT TAA TAC TAG
## uc010nxq.1   6   1   1   2   1   2   3   2   1   3   0   1   5   2   0   1   1
## uc001abv.1   5   2   1   5   1   0   0   6   4   2   0   3   3   1   0   0   0
## uc001abw.1  41   7   7  38  11  11   7  30  26   7   1   9  12   3   0   6   0
##            TAT TCA TCC TCG TCT TGA TGC TGG TGT TTA TTC TTG TTT
## uc010nxq.1   0   0   3   0   3   0   3   0   3   0   3   2   3
## uc001abv.1   0   0   5   0   2   0   4   1   2   0   3   1   0
## uc001abw.1   2   4  17   6   4   1   8   5   4   0  14   5   2
```

The codon coverage can be used to study the ribosome translation dynamics. One can test if codon motifs accumulating ribo-seq reads represent a stalling in the ribosome progression. In the *RiboProfiling* package we have implemented the `codonPCA` function that performs a PCA analysis on a matrix of read density on codons (figures 8).

```
codonUsage <- codonData[[1]]
codonCovMatrix <- codonData[[2]]
#keep only genes with a minimum number of reads
nbrReadsGene <- apply(codonCovMatrix, 1, sum)
ixExpGenes <- which(nbrReadsGene >= 50)
codonCovMatrix <- codonCovMatrix[ixExpGenes, ]
#get the PCA on the codon coverage
codonCovMatrixTransp <- t(codonCovMatrix)
rownames(codonCovMatrixTransp) <- colnames(codonCovMatrix)
colnames(codonCovMatrixTransp) <- rownames(codonCovMatrix)
listPCACodonCoverage <- codonPCA(codonCovMatrixTransp, "codonCoverage")
listPCACodonCoverage[[2]]
```

```
## $`PC_1-2`
##
## $`PC_1-3`
##
## $`PC_1-4`
##
```

```
## $`PC_2-3`
##
## $`PC_2-4`
```

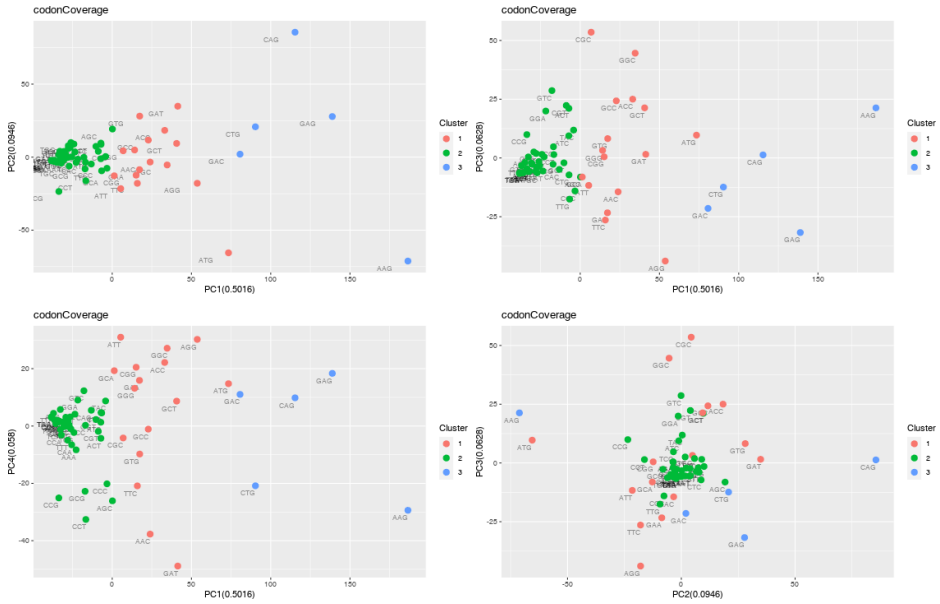

Figure 8: PCA on codon coverage for the ctrlGAlignments example data.

```
sessionInfo()
```

```
## R version 4.1.0 RC (2021-05-10 r80283)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Mojave 10.14.6
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.1/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] stats4    parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
## [1] RSQLite_2.2.7        RiboProfiling_1.22.0 Biostrings_2.60.0
## [4] GenomeInfoDb_1.28.0  XVector_0.32.0       IRanges_2.26.0
## [7] S4Vectors_0.30.0     BiocGenerics_0.38.0  knitr_1.33
##
## loaded via a namespace (and not attached):
##   [1] colorspace_2.0-1
##   [2] rjson_0.2.20
##   [3] ellipsis_0.3.2
##   [4] biovizBase_1.40.0
```

```
##   [5] htmlTable_2.2.1
##   [6] GenomicRanges_1.44.0
##   [7] base64enc_0.1-3
##   [8] dichromat_2.0-0
##   [9] rstudioapi_0.13
##  [10] farver_2.1.0
##  [11] bit64_4.0.5
##  [12] AnnotationDbi_1.54.0
##  [13] fansi_0.4.2
##  [14] sqldf_0.4-11
##  [15] splines_4.1.0
##  [16] ggbio_1.40.0
##  [17] cachem_1.0.5
##  [18] Formula_1.2-4
##  [19] Rsamtools_2.8.0
##  [20] cluster_2.1.2
##  [21] dbplyr_2.1.1
##  [22] png_0.1-7
##  [23] graph_1.70.0
##  [24] BiocManager_1.30.15
##  [25] compiler_4.1.0
##  [26] httr_1.4.2
##  [27] backports_1.2.1
##  [28] assertthat_0.2.1
##  [29] Matrix_1.3-3
##  [30] fastmap_1.1.0
##  [31] lazyeval_0.2.2
##  [32] TxDb.Hsapiens.UCSC.hg19.knownGene_3.2.2
##  [33] htmltools_0.5.1.1
##  [34] prettyunits_1.1.1
##  [35] tools_4.1.0
##  [36] gtable_0.3.0
##  [37] glue_1.4.2
##  [38] GenomeInfoDbData_1.2.6
##  [39] reshape2_1.4.4
##  [40] dplyr_1.0.6
##  [41] rappdirs_0.3.3
##  [42] Rcpp_1.0.6
##  [43] Biobase_2.52.0
##  [44] vctrs_0.3.8
##  [45] nlme_3.1-152
##  [46] rtracklayer_1.52.0
##  [47] xfun_0.23
##  [48] stringr_1.4.0
##  [49] proto_1.0.0
##  [50] lifecycle_1.0.0
##  [51] ensembldb_2.16.0
##  [52] restfulr_0.0.13
##  [53] XML_3.99-0.6
##  [54] zlibbioc_1.38.0
##  [55] scales_1.1.1
```

```
##  [56] BSgenome_1.60.0
##  [57] BiocStyle_2.20.0
##  [58] VariantAnnotation_1.38.0
##  [59] ProtGenerics_1.24.0
##  [60] hms_1.1.0
##  [61] MatrixGenerics_1.4.0
##  [62] RBGL_1.68.0
##  [63] SummarizedExperiment_1.22.0
##  [64] AnnotationFilter_1.16.0
##  [65] RColorBrewer_1.1-2
##  [66] yaml_2.2.1
##  [67] curl_4.3.1
##  [68] memoise_2.0.0
##  [69] gridExtra_2.3
##  [70] ggplot2_3.3.3
##  [71] biomaRt_2.48.0
##  [72] rpart_4.1-15
##  [73] reshape_0.8.8
##  [74] latticeExtra_0.6-29
##  [75] stringi_1.6.2
##  [76] highr_0.9
##  [77] BiocIO_1.2.0
##  [78] checkmate_2.0.0
##  [79] GenomicFeatures_1.44.0
##  [80] filelock_1.0.2
##  [81] BiocParallel_1.26.0
##  [82] chron_2.3-56
##  [83] rlang_0.4.11
##  [84] pkgconfig_2.0.3
##  [85] matrixStats_0.58.0
##  [86] bitops_1.0-7
##  [87] evaluate_0.14
##  [88] lattice_0.20-44
##  [89] purrr_0.3.4
##  [90] labeling_0.4.2
##  [91] GenomicAlignments_1.28.0
##  [92] htmlwidgets_1.5.3
##  [93] bit_4.0.4
##  [94] tidyselect_1.1.1
##  [95] BSgenome.Hsapiens.UCSC.hg19_1.4.3
##  [96] GGally_2.1.1
##  [97] plyr_1.8.6
##  [98] magrittr_2.0.1
##  [99] R6_2.5.0
## [100] magick_2.7.2
## [101] generics_0.1.0
## [102] Hmisc_4.5-0
## [103] DelayedArray_0.18.0
## [104] DBI_1.1.1
## [105] mgcv_1.8-35
## [106] gsubfn_0.7
```

```
## [107] pillar_1.6.1
## [108] foreign_0.8-81
## [109] survival_3.2-11
## [110] KEGGREST_1.32.0
## [111] RCurl_1.98-1.3
## [112] nnet_7.3-16
## [113] tibble_3.1.2
## [114] crayon_1.4.1
## [115] utf8_1.2.1
## [116] OrganismDbi_1.34.0
## [117] BiocFileCache_2.0.0
## [118] rmarkdown_2.8
## [119] jpeg_0.1-8.1
## [120] progress_1.2.2
## [121] grid_4.1.0
## [122] data.table_1.14.0
## [123] blob_1.2.1
## [124] digest_0.6.27
## [125] munsell_0.5.0
## [126] tcltk_4.1.0
```