groHMM Tutorial

November 19, 2021

Contents

1 Introduction

Global Nuclear Run On and Sequencing (GRO-seq) was developed for comprehensively map transcriptional activity in cells [**?**, **?**]. GRO-seq, which provides a genome wide 'map' of the position and orientation of all transcriptionally active RNA polymerases, has become increasingly widely used in recent years because it has numerous advantages compared to alternative methods of transcriptome profiling, such as expression microarrays and RNA-seq. Among these, GRO-seq provides information on instantaneous transactional responses because it detects primary transcription, as opposed to mature, processed mRNA. In addition, because it is independent of RNA polyadenylation, processing, and stability, GRO-seq provides extensive information on the non-coding transcriptome, including primary miRNAs, lincRNAs, enhancer RNAs, and potentially additional, yet undiscovered classes of transcription occurring in cells [**?**, **?**, **?**]. Thus, GRO-seq data provides a complete and instantaneous picture of transcription, and has extensive applications in deciphering the mechanisms of transcriptional regulation.

We have recently developed several important analytical approaches which make use of GRO-seq data to address new biological questions. Our pipleline has been packaged and documented, resulting in the *groHMM* package for Bioconductor. Among the more advanced

features, *groHMM* predicts the boundaries of transcriptional activity across the genome *de novo* using a two-state hidden Markov model (HMM). Our model essentially divides the genome into "transcribed" and "non-transcribed" regions in a strand specific manner [**?**]. We also use HMMs to identify the leading edge of Pol II at genes activated by a stimulus in GRO-seq time course data. This approach allows the genome-wide interrogation of transcription rates in cells [**?**].

In addition to these advanced features, *groHMM* provides wrapper functions for counting raw reads [**?**], generating wiggle files for visualization [**?**], and creating metagene (averaging) plots. *groHMM* takes over all aspects of analysis after reads have been aligned to a reference genome with short-read alignment tools. Although *groHMM* is tailored towards GRO-seq data, the same functions and analytical methodologies can, in principal, be applied to a wide variety of other short read data sets since the package includes a number of easily usable and extensible functions for general short read data analysis. This guide focuses on the most common application of the package.

2 Preparation

The *groHMM* package is available in the Bioconductor and can be downloaded as follows:

```
> if (!requireNamespace("BiocManager", quietly=TRUE))
+     install.packages("BiocManager")
> BiocManager::install("groHMM")
```

The following packages are not required to use *groHMM*, but they are used to download annotations and evaluate transcripts, and should be installed for this tutorial.

```
> BiocManager::install("GenomicFeatures")
> BiocManager::install("org.Hs.eg.db")
> BiocManager::install("edgeR")
> BiocManager::install("TxDb.Hsapiens.UCSC.hg19.knownGene")
```

3 *groHMM* Workflow

3.1 Read GRO-seq Data Files

In this tutorial we will use example data from Hah et al. [2011] (GEO accession GSE27463). This experiment was designed to assess transcriptional changes following treatment of MCF-7 cells with 17β estradiol (E2). The data include a time course of GRO-seq data following treatment with E2 (*i.e.*, 0, 10, 40 and 160 min.). Two biological replicates are available for each time point. Bed files were obtained from GEO and lifted over to hg19 using the UCSC *liftOver* tool. In order to make the package size more manageable, we have included data from chromosome 7 only in 0 and 10 min. conditions. *groHMM* supports parallel processing and the number of cores to use can be set using `mc.cores` option.

```
> library(groHMM)
> options(mc.cores=getCores(4))
```

groHMM uses the *GRanges* class from the *GenomicRanges* packages to represent a collection of genomic features, allowing synergy with other useful packages in Bioconductor. Most of the functions in the package take at least two arguments: 'reads' and 'features'. Reads represent the genomic coordinates of a set of mapped short reads. Features represent a set of genomic coordinates of interest such as genes, exons, or transcripts.

The example data included in this package can be loaded into *R* using the following commands.

```
> S0mR1 <- as(readGAlignments(system.file("extdata", "S0mR1.bam",
+         package="groHMM")), "GRanges")
> S0mR2 <- as(readGAlignments(system.file("extdata", "S0mR1.bam",
+         package="groHMM")), "GRanges") # Use R1 as R2
> S40mR1 <- as(readGAlignments(system.file("extdata", "S40mR1.bam",
+         package="groHMM")), "GRanges")
> S40mR2 <- as(readGAlignments(system.file("extdata", "S40mR1.bam",
+         package="groHMM")), "GRanges") # Use R1 as R2
```

3.2 Create a Wiggle File

Wiggle files are created for each strand after replicates are combined in order to visualize GRO-seq data in the UCSC genome browser. Wiggle files can be also normalized by the sequencing depth, *i.e.*, average number of reads in the dataset. `writeWiggle` function is a wrapper of `export` in *rtracklayer* for generation of wiggle/bigWig type of files.

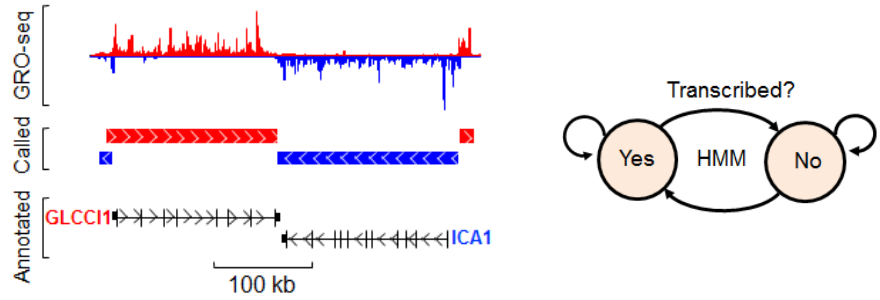
```
> # Combine replicates
> S0m <- c(S0mR1, S0mR2)
> S40m <- c(S40mR1, S40mR2)
```

```
> writeWiggle(reads=S0m, file="S0m_Plus.wig", fileType="wig", strand="+",
+         reverse=FALSE)
> writeWiggle(reads=S0m, file="S0m_Minus.wig", fileType="wig", strand="-",
+         reverse=TRUE)
> # For BigWig file:
> # library(BSgenome.Hsapiens.UCSC.hg19)
> # si <- seqinfo(BSgenome.Hsapiens.UCSC.hg19)
> # writeWiggle(reads=S0m, file="S0m_Plus.wig", fileType="BigWig", strand="+",#
> #               reverse=FALSE, seqinfo=si)
>
> # Normalized wiggle files
> expCounts <- mean(c(NROW(S0m), NROW(S40m)))
> writeWiggle(reads=S0m, file="S0m_Plus_Norm.wig", fileType="wig", strand="+",
+           normCounts=expCounts/NROW(S0m), reverse=FALSE)
```

The resulting wiggle files can be uploaded as 'custom tracks' in the UCSC genome browser, or your visualization software of choice.

3.3 Transcript Calling

In *groHMM*, transcribed regions are detected *de novo* using a two-state hidden Markov model (HMM). The model takes GRO-seq read counts as input across the genome and divides the genome into "transcribed" and "non-transcribed" state as shown in Figure 1. First, a



(a) GRO-seq with called transcripts **(b)** 2-state HMM

Figure 1: **HMM calling of GRO-seq data**

single read set is generated by combining all samples for each time point. This combined approach improves sensitivity for transcripts with low expression levels. Combined reads are used to train the model parameters using the Baum-Welch Expectation Maximization (EM) algorithm. Each strand is modeled separately dividing the genome into non-overlapping 50 bp windows classified as either state.

```
> Sall <- sort(c(S0m, S40m))
> # hmmResult <- detectTranscripts(Sall, LtProbB=-200, UTS=5,
> #                   threshold=1)
> # Load hmmResult from the saved previous run
> load(system.file("extdata", "Vignette.RData", package="groHMM"))
> txHMM <- hmmResult$transcripts
```

```
> head(txHMM)

GRanges object with 6 ranges and 2 metadata columns:
      seqnames          ranges strand |  type            ID
         <Rle>       <IRanges>  <Rle> | <Rle>   <character>
  [1]     chr7   199750-203899      + |    tx  chr7_199750+
  [2]     chr7   561050-568649      + |    tx  chr7_561050+
  [3]     chr7   767600-834149      + |    tx  chr7_767600+
  [4]     chr7   852400-941849      + |    tx  chr7_852400+
  [5]     chr7 1176600-1178799      + |    tx chr7_1176600+
  [6]     chr7 1198200-1210799      + |    tx chr7_1198200+
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

The `detectTranscripts` function also uses two hold-out parameters. These parameters, specified by the arguments `LtProbB` and `UTS`, represents the log-transformed transition probability of switching from transcribed state to non-transcribed state and variance of the emission probability for reads in the non-transcribed state, respectively. Holdout parameters are used to optimize the performance of HMM predictions on known genes.

3.4 Evaluation of Transcript Calling

Predicted transcripts are evaluated by comparison to known annotations, making the assumption that GRO-seq transcripts should largely be in agreement with available annotations. Two types of error may occur, as described below. The HMM parameters are evaluated by the sum of the error rates. The procedure involves collecting a set of high-confidence reference transcripts. An annotation dataset can be constructed by downloading from the UCSC database or alternatively, pre-made TranscriptDb objects can be used if they are available in the Bioconductor. We will use the UCSC knownGene track and retrieve transcript annotations with *GenomicFeatures* [**?**] package.

```
> library(TxDb.Hsapiens.UCSC.hg19.knownGene)
> kgdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
> library(GenomicFeatures)
> # For refseq annotations:
> # rgdb <- makeTxDbFromUCSC(genome="hg19", tablename="refGene")
> # saveDb(hg19RGdb, file="hg19RefGene.sqlite")
> # rgdb <- loadDb("hg19RefGene.sqlite")
> kgChr7 <- transcripts(kgdb, filter=list(tx_chrom = "chr7"),
+                       columns=c("gene_id", "tx_id", "tx_name"))
> seqlevels(kgChr7) <- seqlevelsInUse(kgChr7)
```

Because annotations do not provide precise cell type-specific expression information, overlapping transcripts must be merged into a single set, in which each annotation represents the 5′- and 3′-most boundaries of genes. Different isoforms of each gene are collapsed into one using the ENTREZID. These consensus annotations are used for the evaluation of HMM calling.

```
> # Collapse overlapping annotations
> kgConsensus <- makeConsensusAnnotations(kgChr7, keytype="gene_id",
+         mc.cores=getOption("mc.cores"))
> library(org.Hs.eg.db)
> map <- select(org.Hs.eg.db,
+     keys=unlist(mcols(kgConsensus)$gene_id),
+         columns=c("SYMBOL"), keytype=c("ENTREZID"))
> mcols(kgConsensus)$symbol <- map$SYMBOL
> mcols(kgConsensus)$type <- "gene"
```

There are two types of error that can be evaluated when comparing predicted transcripts with annotations. (1) The number of transcripts overlapping two or more annotations, (*i.e.*, these transcripts 'merged annotations together') and (2) the number of annotations that overlap two or more transcripts on the same strand (*i.e.*, we say that these transcript calls 'dissociated a single annotation') must be determined. The optimal tuning parameters can be found by minimizing the sum of the two errors. This approach allows identification of the model that best fits the existing annotations and should more precisely predict transcripts in non-annotated parts of the genome.

```
> e <- evaluateHMMInAnnotations(txHMM, kgConsensus)
> e$eval

  merged dissociated total  errorRate txSize
1     64          47   111 0.06098901    830
```

3.5 HMM Tuning

Here we demonstrate how the optimal value for each tuning parameters can be obtained by running HMM multiple times over a certain range of the parameters. Among the nine test cases, both the sum of errors and error rate per called transcript show minimal at case #7, as shown below. The variation of errors should be greater if whole chromosomes are used. Also, a larger set of the parameters might be used in practice. This tunning step takes long time, so you may skip it for quick review of the package.

```
> tune <- data.frame(LtProbB=c(rep(-100,3), rep(-200,3), rep(-300,3)),
+            UTS=rep(c(5,10,15), 3))
> Fp <- windowAnalysis(Sall, strand="+", windowSize=50)
> Fm <- windowAnalysis(Sall, strand="-", windowSize=50)
> # evals <- mclapply(seq_len(9), function(x) {
> #        hmm <- detectTranscripts(Fp=Fp, Fm=Fm, LtProbB=tune$LtProbB[x],
> #                  UTS=tune$UTS[x])
> #        e <- evaluateHMMInAnnotations(hmm$transcripts, kgConsensus)
> #        e$eval
> #        }, mc.cores=getOption("mc.cores"),  mc.silent=TRUE)
> tune <- cbind(tune, do.call(rbind, evals))  # evals from the previous run
```

```
> tune
```

```
  LtProbB UTS merged dissociated total  errorRate txSize
1    -100   5     50         135   185 0.07769845   1391
2    -100  10     61         177   238 0.07775237   2071
3    -100  15     68         201   269 0.07441217   2625
4    -200   5     64          47   111 0.06098901    830
5    -200  10     74          65   139 0.06547339   1133
6    -200  15     80          76   156 0.06643952   1358
7    -300   5     69          22    91 0.05501814    664
8    -300  10     82          30   112 0.06005362    875
9    -300  15     90          41   131 0.06498016   1026
```

```
> which.min(tune$total)
```

```
[1] 7
```

```
> which.min(tune$errorRate)
```

```
[1] 7
```

To robustly compare transcripts with known genes, densities representing the frequency of transcripts can be calculated relative to their mapped gene annotations. Conceptually, the plot is divided into three distinct regions as shown in Figure 2, including upstream of known gene annotations, inside genes, and downstream of the annotated polyadenylation site.

Metrics to evaluate the degree of overlap with gene annotations focus on the region upstream of gene annotations, which provides a measure of specificity, and the region inside of genes, which provides a measure of sensitivity. The region downstream of the polyadenylation site is known to contain residual transcription [**?**] and consequently is not used to define quality.

The metrics are defined relative to an 'ideal' transcript caller, which takes the form of a step function (*i.e.*, red line in the plot). Our quality metrics represent the following (see the plot below for a graphical representation):

1. true positive; gene body (TP) = (gene annotation area under the curve) /(max area for matched transcripts),
2. false negative; gene body (FN) = 1 - TP,
3. 5′ false positive; upstream region (5′FP) = (5′ overhang area under the curve) /(max area for matched transcripts),
4. 5′ true negative; upstream region (5′TN) = TP - 5′FP.

We constrained 5′FP and 5′TN so that their sum to be TP in order to use the upstream region only if positive number of transcipts are called in the gene body for the calucation of the quality metrics. Note that these quality metrics are conceptually very similar to true positive, true negative, false positive and false negative, respectively. During the comparison, only expressed annotations are used and genes either too small or too large in size are excluded. And also size of transcripts and annotations are scaled to a smaller unit, *i.e.*, 1K for a visual representation. Here best overlapped annotations or transcripts are used for either 'merged annotations' or 'dissociated a single annotation' type of error. Final quality metrics are represented as TUA (Transcription Unit Accuracy).

```
> getExpressedAnnotations <- function(features, reads) {
+      fLimit <- limitToXkb(features)
+      count <- countOverlaps(fLimit, reads)
+      features <- features[count!=0,]
+      return(features[(quantile(width(features), .05) < width(features))
+     & (width(features) < quantile(width(features), .95)),])}
> conExpressed <- getExpressedAnnotations(features=kgConsensus,reads=Sall)
```

```
> td <- getTxDensity(txHMM, conExpressed, mc.cores=getOption("mc.cores"))

Merged annotations:  51
Dissociated a single annotation:  35
Overlaps between transcript and annotation:
Total =  494 Used for density =  389

> u <- par("usr")
> lines(c(u[1], 0, 0, 1000, 1000, u[2]), c(0,0,u[4]-.04,u[4]-.04,0,0),
+       col="red")
> legend("topright", lty=c(1,1), col=c(2,1), c("ideal", "groHMM"))
> text(c(-500,500), c(.05,.5), c("FivePrimeFP", "TP"))
> td

$FivePrimeFP
[1] 0.1175103

$TP
[1] 0.7902918

$PostTTS
[1] 0.3441388

$TUA
[1] 0.8172262
```

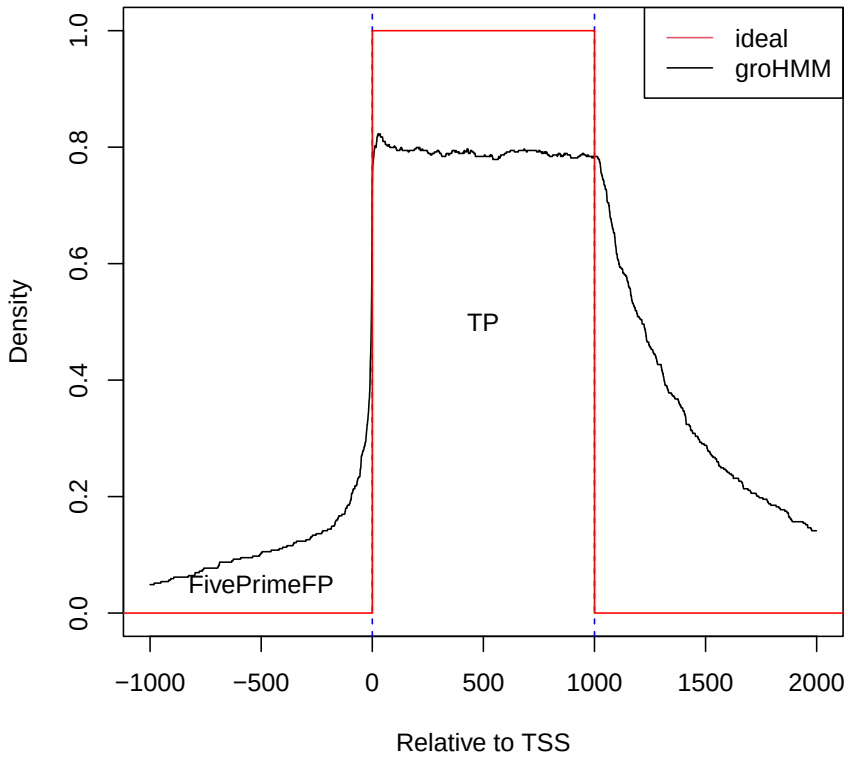


Figure 2: Transcript Density Plot

3.6 Working with non-mammalian Genomes

If your target of study is non-mammalian, you can retrieve the relevant annotations from the Bioconductor if they are supported. You can check the availability with the function `supportedUCSCtable` in *GenomicsFeautes*. If the organism is not supported, you can still build consensus annotations by directly downloading the annotation table for the organism from the UCSC genome browser. The following command line shows the mysql query in linux to download protein-coding RefSeq genes for all chromosomes except chrM for C. Elegans saving into refgene.bed file. You can find more information about using MySQL for the UCSC genome browser at https://genome.ucsc.edu/goldenPath/help/mysql.html.

```
> # mysql --user=genome --host=genome-mysql.cse.ucsc.edu ce10 -e \
> #  "select chrom, txStart, txEnd, name, exonCount, strand, name2 from refGene \
> #   where chrom not like chrom!='chrM' and cdsStart != cdsEnd" | tail -n +1 > refgene.bed
>
> # G <- read.table("refgene.bed", header=TRUE, stringsAsFactors=FALSE, sep="\t")
> # ce <- GRanges(G$chrom, IRanges(G$txStart, G$txEnd), strand=G$strand, \
> #               access=G$name, gene_id=G$name2)
> # ceConsensus <- makeConsensusAnnotations(ce, keytype="gene_id", \
> #                mc.cores=getOption("mc.cores"))
```

As for transcript calling, the default values for `detectTranscripts` were set for mammalian genome. In case of C. Elegans genome, it is much smaller than human genome and the genes are more tightly located. So we recommend to explore higher values for the transition probability from the transcribed to non-transcribed state. For example, you can use i.e., values >-50 instead of -200 for `LtProbB`.

3.7 Repairing Transcript Calling with Annotations

Prediction of transcripts by the HMM is not perfect. Discrepancies with the annotations will occur even after the parameters are optimally tuned. Transcript calls can be 'fixed' for known types of error by (1) breaking transcripts that have merged annotations and (2) combining transcripts that have dissociated a single annotation. The following method will generate a final set of transcripts for further analysis.

```
> bPlus <- breakTranscriptsOnGenes(txHMM, kgConsensus, strand="+")

19  transcripts are broken into  46

> bMinus <- breakTranscriptsOnGenes(txHMM, kgConsensus, strand="-")

14  transcripts are broken into  32

> txBroken <- c(bPlus, bMinus)
> txFinal <- combineTranscripts(txBroken, kgConsensus)

87  transcripts are combined to  34
```

```
> tdFinal <- getTxDensity(txFinal, conExpressed, mc.cores=getOption("mc.cores"))
```

3.8 Differential Analysis with *edgeR*

There are several packages in Bioconductor for differential expression analysis such as *DESeq*, *baySeq*, *DEGSeq*, or *edgeR*. *edgeR* [**?**] is used for this demonstration. Differential expression analysis can be done using either by called transcripts or known annotations depending on the nature of your research question. The procedures are quite similar except reads are counted using the genomic locations of either called transcript or known annotations. For longer transcripts or annotated genes, we use a window of $+1$ to $+13$ kb from the transcription start site (TSS), which was chosen in order to exclude reads from RNA polymerases engaged at the promoter and to allow enough time for the elongation of newly initiated Pol II (See Hah et al., 2011). However, variations can be used, including the entire length of the called or annotated transcripts.

```
> # For called transcripts
> library(edgeR)
> txLimit <- limitToXkb(txFinal)
> ctS0mR1 <- countOverlaps(txLimit, S0mR1)
> ctS0mR2 <- countOverlaps(txLimit, S0mR2)
> ctS40mR1 <- countOverlaps(txLimit, S40mR1)
> ctS40mR2 <- countOverlaps(txLimit, S40mR2)
> pcounts <- as.matrix(data.frame(ctS0mR1, ctS0mR2, ctS40mR1, ctS40mR2))
> group <- factor(c("S0m", "S0m", "S40m", "S40m"))
> lib.size <- c(NROW(S0mR1), NROW(S0mR2), NROW(S40mR1), NROW(S40mR2))
> d <- DGEList(counts=pcounts, lib.size=lib.size, group=group)
```

```
> d <- estimateCommonDisp(d)
> et <- exactTest(d)
> de <- decideTestsDGE(et, p=0.001, adjust="fdr")
> detags <- seq_len(NROW(d))[as.logical(de)]
> # Number of transcripts regulated at 40m
> cat("up: ",sum(de==1), " down: ", sum(de==-1), "\n")
```

```
up:  186  down:  152
```

```
> plotSmear(et, de.tags=detags)
> # 2 fold up or down
> abline(h = c(-1,1), col="blue")
```

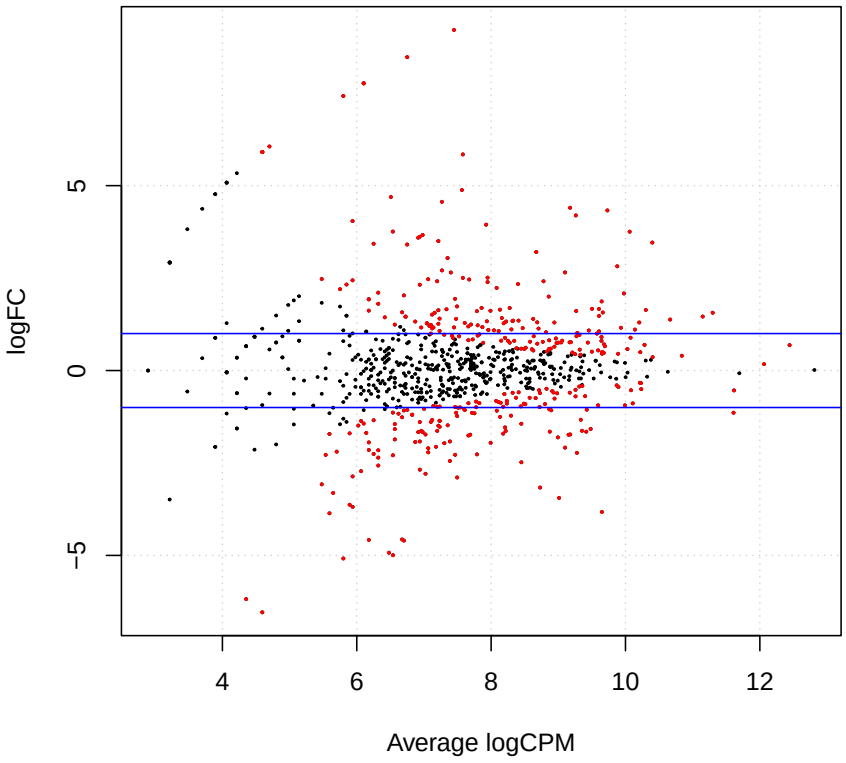


Figure 3: Transcript MAplot

```
> # For ucsc knownGenes
> kgChr7 <- transcripts(kgdb, filter <- list(tx_chrom = "chr7"),
+                       columns=c("gene_id", "tx_id", "tx_name"))
> map <- select(org.Hs.eg.db,
+         keys=unique(unlist(mcols(kgChr7)$gene_id)),
+         columns=c("SYMBOL"), keytype=c("ENTREZID"))
> missing <- elementNROWS(mcols(kgChr7)[,"gene_id"]) == 0
```

```
> kgChr7 <- kgChr7[!missing,]
> inx <- match(unlist(mcols(kgChr7)$gene_id), map$ENTREZID)
> mcols(kgChr7)$symbol <- map[inx,"SYMBOL"]
> kgLimit <- limitToXkb(kgChr7)
> ctS0mR1 <- countOverlaps(kgLimit, S0mR1)
> ctS0mR2 <- countOverlaps(kgLimit, S0mR2)
> ctS40mR1 <- countOverlaps(kgLimit, S40mR1)
> ctS40mR2 <- countOverlaps(kgLimit, S40mR2)
> counts <- as.matrix(data.frame(ctS0mR1, ctS0mR2, ctS40mR1, ctS40mR2))
> group <- factor(c("S0m", "S0m", "S40m", "S40m"))
> lib.size <- c(NROW(S0mR1), NROW(S0mR2), NROW(S40mR1), NROW(S40mR2))
> d <- DGEList(counts=counts, lib.size=lib.size, group=group)
> d <- estimateCommonDisp(d)
> et <- exactTest(d)
> de <- decideTestsDGE(et, p=0.001, adjust="fdr")
> detags <- seq_len(NROW(d))[as.logical(de)]
> symbols <- mcols(kgChr7)$symbol
> # Number of unique genes regulated at 40m
> cat("up: ", NROW(unique(symbols[de==1])), "\n")

up:  149

> cat("down: ", NROW(unique(symbols[de==-1])), "\n")

down:  111

> plotSmear(et, de.tags=detags)
> abline(h = c(-1,1), col="blue")
```

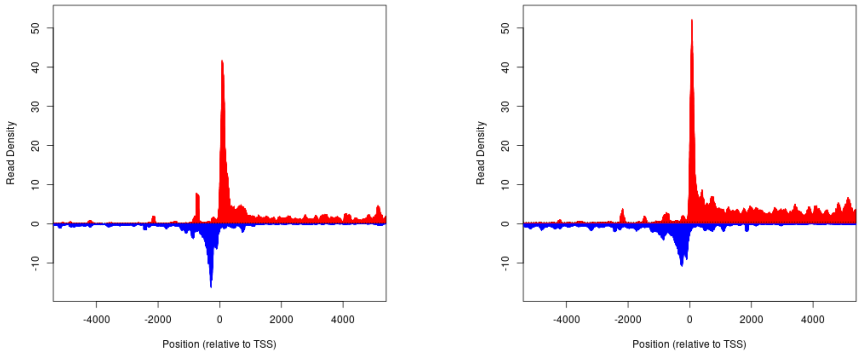
3.9 Metagene Analysis

Metagenes show the distribution of reads near TSS of a set of regulated genes (or some other alignable genomic features of interest). It can be thought as a smoothed average of read density weighted by expression over the set of TSS. The `runMetaGene` function has option for sampling. If `TRUE`, 10% of the transcription units are sampled with replacement 1,000 times and median value at each position in the transcription unit over the samples is used for final metagene result. Using subsampling results in an image is more robust to outliers, especially when the size of sample is relatively small.

```
> upGenes <- kgChr7[de==1,]
> expReads <- mean(c(NROW(S0m), NROW(S40m)))
> # Metagene around TSS
> mg0m <- runMetaGene(features=upGenes, reads=S0m, size=100,
+                         normCounts=expReads/NROW(S0m), sampling=FALSE,
+                mc.cores=getOption("mc.cores"))
> mg40m <- runMetaGene(features=upGenes, reads=S40m, size=100,
+                          normCounts=expReads/NROW(S40m), sampling=FALSE,
+                 mc.cores=getOption("mc.cores"))
```

```
> plotMetaGene <- function(POS=c(-10000:+9999), mg, MIN, MAX){
+     plot(POS, mg$sense, col="red", type="h", xlim=c(-5000, 5000),
+         ylim=c(floor(MIN),ceiling(MAX)), ylab="Read Density",
```

```
+         xlab="Position (relative to TSS)")
+      points(POS, (-1*rev(mg$antisense)), col="blue", type="h")
+      abline(mean(mg$sense[5000:8000]), 0, lty="dotted")
+ }
> MAX <- max(c(mg0m$sense, mg40m$sense))
> MIN <- -1*max(c(mg0m$antisense, mg40m$antisense))
> plotMetaGene(mg=mg0m, MIN=MIN, MAX=MAX)
> plotMetaGene(mg=mg40m, MIN=MIN, MAX=MAX)
```



(a) 0 min.

(b) 40 min.

Figure 4: Metagenes

4 Session Info

```
> toLatex(sessionInfo())
```

- R Under development (unstable) (2021-11-10 r81171), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_GB`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Running under: `Ubuntu 20.04.3 LTS`
- Matrix products: default
- BLAS: `/home/biocbuild/bbs-3.15-bioc/R/lib/libRblas.so`
- LAPACK: `/home/biocbuild/bbs-3.15-bioc/R/lib/libRlapack.so`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: AnnotationDbi 1.57.1, Biobase 2.55.0, BiocGenerics 0.41.2, Biostrings 2.63.0, GenomeInfoDb 1.31.1, GenomicAlignments 1.31.2, GenomicFeatures 1.47.2, GenomicRanges 1.47.5, IRanges 2.29.1, MASS 7.3-54, MatrixGenerics 1.7.0, Rsamtools 2.11.0, S4Vectors 0.33.4,

SummarizedExperiment 1.25.2, TxDb.Hsapiens.UCSC.hg19.knownGene 3.2.2, XVector 0.35.0, edgeR 3.37.0, groHMM 1.29.1, limma 3.51.0, matrixStats 0.61.0, org.Hs.eg.db 3.14.0, rtracklayer 1.55.0

- Loaded via a namespace (and not attached): BiocFileCache 2.3.0, BiocIO 1.5.0, BiocManager 1.30.16, BiocParallel 1.29.2, BiocStyle 2.23.0, DBI 1.1.1, DelayedArray 0.21.2, GenomeInfoDbData 1.2.7, KEGGREST 1.35.0, Matrix 1.3-4, R6 2.5.1, RCurl 1.98-1.5, RSQLite 2.2.8, Rcpp 1.0.7, XML 3.99-0.8, assertthat 0.2.1, biomaRt 2.51.1, bit 4.0.4, bit64 4.0.5, bitops 1.0-7, blob 1.2.2, cachem 1.0.6, compiler 4.2.0, crayon 1.4.2, curl 4.3.2, dbplyr 2.1.1, digest 0.6.28, dplyr 1.0.7, ellipsis 0.3.2, evaluate 0.14, fansi 0.5.0, fastmap 1.1.0, filelock 1.0.2, generics 0.1.1, glue 1.5.0, grid 4.2.0, hms 1.1.1, htmltools 0.5.2, httr 1.4.2, knitr 1.36, lattice 0.20-45, lifecycle 1.0.1, locfit 1.5-9.4, magrittr 2.0.1, memoise 2.0.0, pillar 1.6.4, pkgconfig 2.0.3, png 0.1-7, prettyunits 1.1.1, progress 1.2.2, purrr 0.3.4, rappdirs 0.3.3, restfulr 0.0.13, rjson 0.2.20, rlang 0.4.12, rmarkdown 2.11, stringi 1.7.5, stringr 1.4.0, tibble 3.1.6, tidyselect 1.1.1, tools 4.2.0, utf8 1.2.2, vctrs 0.3.8, xfun 0.28, xml2 1.3.2, yaml 2.2.1, zlibbioc 1.41.0