# *groHMM* Tutorial

Minho Chae Charles G. Danko W. Lee Kraus

May 3, 2013

## 1 Introduction

Global Nuclear Run On and Sequencing (GRO-Seq) was developed for comprehensively map transcriptional activity in cells [Core et al., 2008, Hah et al., 2011]. GRO-seq, which provides a genome wide 'map' of the position and orientation of all transcriptionally active RNA plolymerases, has become increasingly widely used in recent years becuase it has numerous advantageous compared to alternative methods of transcriptome profiling, such as expression microarrays and RNA-seq. Among these, GRO-seq provides information on instanteneous 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 [Hah et al., 2011]. 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 he 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 [Hah et al., 2011]. 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 [Danko et al., 2013].

In addtion to these advanced features, *groHMM* provides basic functionality such as 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 [Gentleman et al., 2004] and can be downloaded as follows:

```
> source("http://bioconductor.org/biocLite.R")
> biocLite("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.

```
> biocLite("GenomicFeatures")
> biocLite("org.Hs.eg.db")
> biocLite("edgeR")
```

# 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 17$\beta$ estradiol (E2). The data include a time ccurse 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.

```
> library(groHMM)
```

*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.

```
> load(system.file("extdata", "hah.chr7.RData", package="groHMM"))
```

## 3.2 Create a Wiggle File

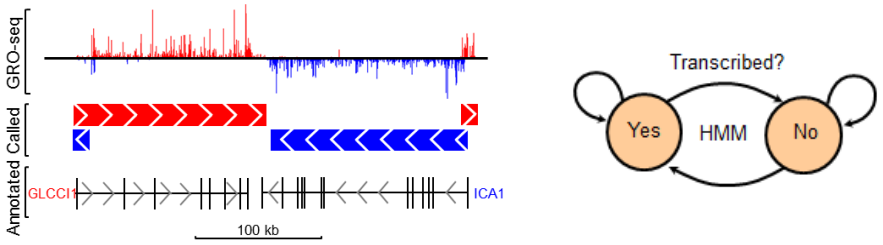
Wiggle files are created for each strand after replicates are combined in order to visualize GRO-seq data in the UCSC genome browser. Wiggle files can also be normalized by the sequencing death, *i.e.*, average number of reads in the dataset.

```
> S0m <- sort(c(S0mR1, S0mR2), decreasing=FALSE)
> S10m <- sort(c(S10mR1, S10mR2), decreasing=FALSE)
> S40m <- sort(c(S40mR1, S40mR2), decreasing=FALSE)
> S160m <- sort(c(S160mR1, S160mR2), decreasing=FALSE)
```

```
> writeWiggle(reads=S0m, file="S0m_Plus", strand="+", size=25,
+             reverse=FALSE)
> writeWiggle(reads=S0m, file="S0m_Minus", strand="-", size=25,
+             reverse=TRUE)
> # Normalized wiggle files
> expCounts <- mean(c(NROW(S0m), NROW(S10m), NROW(S40m), NROW(S160m)))
> writeWiggle(reads=S0m, file="S0m_Plus_Norm", strand="+", size=25,
+            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




(a) GRO-seq with called transcripts

(b) 2-state HMM

Figure 1: HMM calling of GRO-seq data

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 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, S10m, S40m, S160m), decreasing=FALSE)
> hmmResult <- detectTranscripts(Sall, LtProbB=-200, UTS=5,
+                                threshold=1)
> txHMM <- hmmResult$transcripts

> head(txHMM)

GRanges with 6 ranges and 2 metadata columns:
      seqnames              ranges strand |  type            ID
         <Rle>           <IRanges>  <Rle> | <Rle>   <character>
  [1]     chr7 [ 149250,  151349]      + |    tx  chr7_149250+
  [2]     chr7 [ 196150,  231399]      + |    tx  chr7_196150+
  [3]     chr7 [ 558050,  568649]      + |    tx  chr7_558050+
  [4]     chr7 [ 767600,  941699]      + |    tx  chr7_767600+
```

```
  [5]     chr7 [1126400, 1130799]      + |    tx chr7_1126400+
  [6]     chr7 [1170250, 1211449]      + |    tx chr7_1170250+
  ---
  seqlengths:
   chr7
     NA
```

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 RefSeq genes from the UCSC database. The user is not expected to execute this every time; instead, we suggest that you save annotations once and read them whenever needed.

```
> library(GenomicFeatures)
> rgdb <- makeTranscriptDbFromUCSC(genome="hg19", tablename="refGene")
> # Save it for furture use
> # saveDb(hg19RGdb, file="hg19RefGene.sqlite")
> # rgdb <- loadDb("hg19RefGene.sqlite")
> rgChr7 <- transcripts(rgdb, vals <- list(tx_chrom = "chr7"),
+                       columns=c("gene_id", "tx_id", "tx_name"))
> seqlevels(rgChr7) <- "chr7"
```

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 collpsed into one using the ENTREZID. These consensus annotations are used for the evaluation of HMM calling.

```
> # merge overlapping annotations
> rgConsensus <- makeConsensusAnnotations(rgChr7, keytype="gene_id")
> library(org.Hs.eg.db)
> map <- select(org.Hs.eg.db,
+         keys=as.integer(unlist(elementMetadata(rgConsensus)$gene_id)),
+         cols=c("SYMBOL"), keytype=c("ENTREZID"))
> elementMetadata(rgConsensus)$symbol <- map$SYMBOL
> elementMetadata(rgConsensus)$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 'run genes 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 'break up 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 <- evaluateHMM(txHMM, rgConsensus)
```

```
[1] "======================================"
[1] "HMM Errors"
[1] "======================================"
[1] "Run genes together: 116"
[1] "Broken up a single annotation: 57"
[1] "======================================"
[1] "Overlap quality (Median Mean)"
[1] "======================================"
[1] "Tx length: 43200 71990"
[1] "Gene length: 33110 103100"
[1] "Tx overlap: 0.676 0.5974"
[1] "Gene overlap: 1 0.7585"
[1] "Overbases: 16960 41290"
[1] "Similarity: 0.3788 0.4217"
```

## 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, the sum of errors show minimal at case #4 and #7, as shown below. #4 is chosen over #7 for further analysis due to the lower error rate per called transcript. 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),  runGenes=rep(0,9),
+            brokenUp=rep(0,9), totalError=rep(0,9), txSize=rep(0,9),
+            errorRate=rep(0,9))
> for (i in 1:9) {
+     hmm <- detectTranscripts(Sall, LtProbB=tune[i,"LtProbB"],
+                              UTS=tune[i,"UTS"], thresh=1)
+     tx <- hmm$transcripts
+     eh <- evaluateHMM(tx, rgConsensus)
+     tune[i,"runGenes"] <- eh$runGeneError
+     tune[i,"brokenUp"] <- eh$brokenError
+     tune[i,"totalError"] <- eh$runGeneError + eh$brokenError
+     tune[i,"txSize"] <- NROW(tx)
+     tune[i,"errorRate"] <- tune[i,"totalError"] / tune[i,"txSize"]
+ }
>
```

```
> tune
```

```
  LtProbB UTS runGenes brokenUp totalError txSize errorRate
1    -100   5       93      299        392   1734 0.2260669
2    -100  10      103      491        594   2378 0.2497897
3    -100  15      108      603        711   2850 0.2494737
4    -200   5      115       57        172   1084 0.1586716
5    -200  10      125      133        258   1441 0.1790423
6    -200  15      132      174        306   1677 0.1824687
7    -300   5      135       33        168    902 0.1862528
8    -300  10      143       60        203   1144 0.1774476
9    -300  15      150       88        238   1339 0.1777446
```

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

```
[1] 7
```

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

```
[1] 4
```

To robustly compare transcripts with known genes, densities representing the frequency of transcripts are stacked together relative to 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 [Core et al., 2008] 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 transcript density (TTD) = (gene annotation area under the curve) /(max area for matched transcripts),
2. true non-trancript density (TND) = 1 - TTD,
3. false transcript density (FTD) = ($5'$ overhang area under the curve) /(max area for matched trasncripts),
4. false non-transcript density (FND) = 1- FTD.

Note that these quality metrics are conceptually very similar to true positive, true negative, false positive and false negative, respectively. During the comparison, annotations which have expression are used but genes either tool small or too large in size are excluded. And also size of transcripts and annotations are scaled to median size of genes, *i.e.*, 30K. Here best overlapped annotations where a transcript 'run genes together' are used and also best overlapped transcripts among transcripts which broke a single annotation are used. Final quality metrics are represented as an AUC number.

```
> getExpressedAnnotation <- function(features, reads) {
+      fLimit <- limitToXkb(features)
+      count <- countReadsInInterval(features=fLimit, reads=reads)
+      features <- features[count!=0,]
+      return(features[(quantile(width(features), .05)< width(features))
+         & (width(features) < quantile(width(features), .95)),])}
> rgExpressed <- getExpressedAnnotation(features=rgConsensus,reads=Sall)

> plotTxDensity(txHMM, rgExpressed, scale=TRUE, runGenes="best",
+               brokenAnnotation="best", pBar=FALSE)

[1] "(+)  Run genes together: 55"
[1] "(+)  Broken up a single annotation: 18"
[1] "(-)  Run genes together: 38"
[1] "(-)  Broken up a single annotation: 18"

[1] "FTD: 0.13 TTD: 0.82 PostTTS: 0.36 AUC: 0.84"
[1] "No of matched transcripts: 455"

> u <- par("usr")
> lines(c(u[1], 0, 0, 30000, 30000, u[2]), c(0,0,u[4]-10,u[4]-10,0,0),
+       col="red")
> legend("topright",lty=c(1,1), col=c(2,1), c("ideal", "groHMM"))
> text(c(-15000,15000), c(15,210), c("FTD", "TTD"))
```

### 3.6 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 run together and (2) combining transcripts that have been broken. The following method will generate a final set of transcripts for further analysis.

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

[1] "Initial transcripts: 552"
[1] "26 transcripts are broken into 65"
[1] "Final transcripts: 591"

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

[1] "Initial transcripts: 532"
[1] "18 transcripts are broken into 41"
[1] "Final transcripts: 555"

> txBroken <- c(bPlus, bMinus)
> txFinal <- combineTranscripts(txBroken, rgConsensus, debug=FALSE)

[1] "Initial transcripts: 1146"
[1] "56 transcripts are combined to 24"
[1] "Final transcripts: 1114"
```

```
[1] "(+)  Run genes together: 55"
[1] "(+)  Broken up a single annotation: 18"
[1] "(-)  Run genes together: 38"
[1] "(-)  Broken up a single annotation: 18"

[1] "FTD: 0.13 TTD: 0.82 PostTTS: 0.36 AUC: 0.84"
[1] "No of matched transcripts: 455"
```

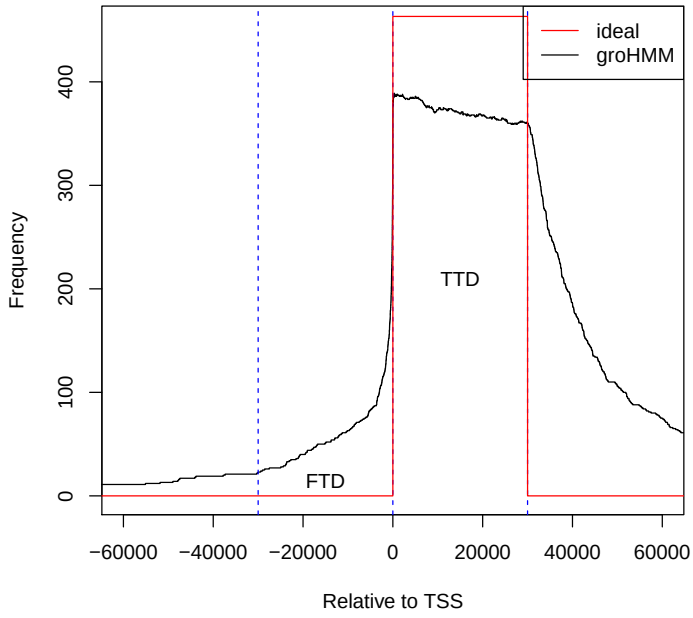



Figure 2: Transcript Density Plot

```
> plotTxDensity(txFinal, rgExpressed, scale=TRUE, runGenes="best",
+               brokenAnnotation="best", pBar=FALSE)
```

```
[1] "(+)  Run genes together: 16"
[1] "(+)  Broken up a single annotation: 9"
[1] "(-)  Run genes together: 16"
[1] "(-)  Broken up a single annotation: 9"

[1] "FTD: 0.09 TTD: 0.85 PostTTS: 0.37 AUC: 0.88"
[1] "No of matched transcripts: 516"
```

## 3.7 Differential Analysis with *edgeR*

There are several packages in Bioconductor for differential expression analysis such as *DESeq*, *baySeq*, *DEGSeq*, or *edgeR*. *edgeR* [Robinson et al., 2010] 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 transcripts
> library(edgeR)
> txLimit <- limitToXkb(txFinal)
> ctS0mR1 <- countReadsInInterval(features=txLimit, reads=S0mR1)
> ctS0mR2 <- countReadsInInterval(features=txLimit, reads=S0mR2)
> ctS40mR1 <- countReadsInInterval(features=txLimit, reads=S40mR1)
> ctS40mR2 <- countReadsInInterval(features=txLimit, reads=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 <- (1:NROW(d))[as.logical(de)]
> # Number of Transcripts regulated at 40m
> print(paste("up:",sum(de==1)))
```

```
[1] "up: 97"
```

```
> print(paste("down:",sum(de==-1)))
```

```
[1] "down: 77"
```

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

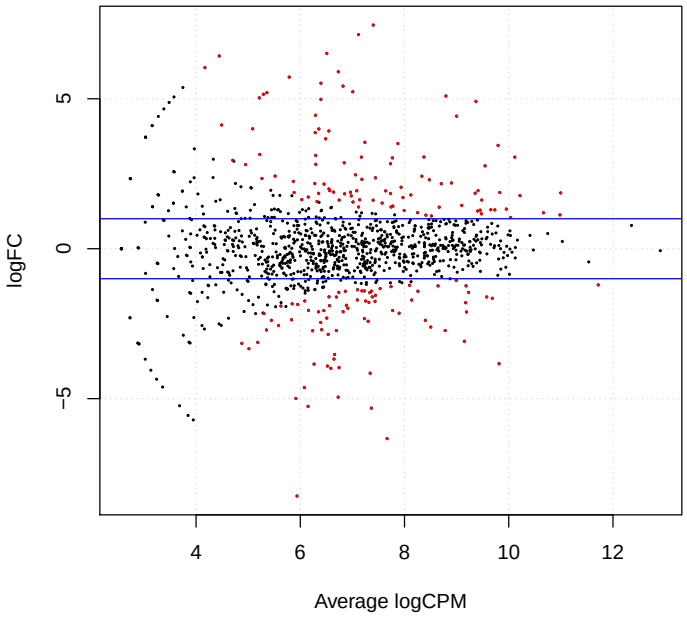


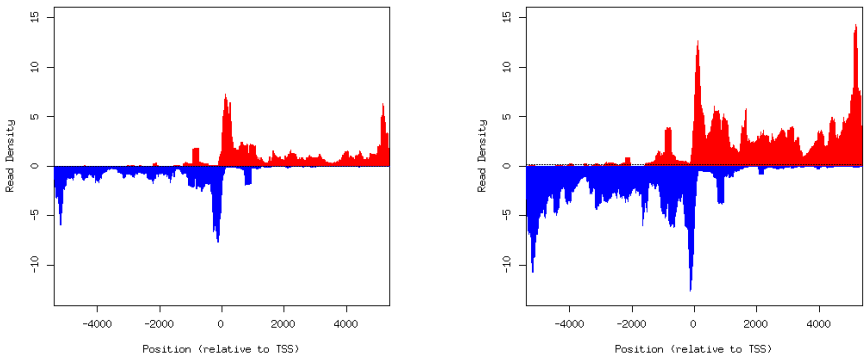

Figure 3: Transcript MAplot

```
> # for refSeq genes
> rgChr7 <- transcripts(rgdb, vals <- list(tx_chrom = "chr7"),
+                       columns=c("gene_id", "tx_id", "tx_name"))
> map <- select(org.Hs.eg.db,
+         keys=as.integer(unique(unlist(elementMetadata(rgChr7)$gene_id))),
+         cols=c("SYMBOL"), keytype=c("ENTREZID"))
> inx <- match(unlist(elementMetadata(rgChr7)$gene_id), map$ENTREZID)
> elementMetadata(rgChr7)$symbol <- map[inx,"SYMBOL"]
> rgLimit <- limitToXkb(rgChr7)
> ctS0mR1 <- countReadsInInterval(features=rgLimit, reads=S0mR1)
> ctS0mR2 <- countReadsInInterval(features=rgLimit, reads=S0mR2)
> ctS40mR1 <- countReadsInInterval(features=rgLimit, reads=S40mR1)
> ctS40mR2 <- countReadsInInterval(features=rgLimit, reads=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 <- (1:NROW(d))[as.logical(de)]
> symbols <- elementMetadata(rgChr7)$symbol
> # Number of Unique Genes at 40m
> print(paste("up:", NROW(unique(unique(symbols[de==1])))))

[1] "up: 56"

> print(paste("down:", NROW(unique(unique(symbols[de==-1])))))

[1] "down: 34"

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

## 3.8 Metagene Analysis

Metagenes show the distribution of reads near TSS of a set of regulated genes (or some other alignalbe 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 that is more robust to outliers, especially when the size of sample is relatively small.

```
> upGenes <- rgChr7[de==1,]
> TSS <- resize(upGenes, width=1)
> expReads <- mean(c(NROW(S0m), NROW(S10m), NROW(S40m), NROW(S160m)))
> count0m <- runMetaGene(features=TSS, reads=S0m, size=100,
+                        normCounts=expReads, sampling=TRUE)
> count40m <- runMetaGene(features=TSS, reads=S40m, size=100,
+                         normCounts=expReads, sampling=TRUE)
```




(a) 0 min.

(b) 40 min.

Figure 4: Metagenes

```
> plotMetaGene <- function(POS=c(-10000:+10000), counts, MIN, MAX){
+     plot(POS, counts$sense, col="red", type="h", xlim=c(-5000, 5000),
+         ylim=c(floor(MIN),ceiling(MAX)), ylab="Read Density",
+         xlab="Position (relative to TSS)")
+      points(POS, (-1*rev(counts$antisense)), col="blue", type="h")
+      abline(mean(counts$sense[5000:8000]), 0, lty="dotted")
+ }
> MAX <- max(c(count0m$sense, count40m$sense))
> MIN <- -1*max(c(count0m$antisense, count40m$antisense))
> plotMetaGene(counts=count0m, MIN=MIN, MAX=MAX)
> plotMetaGene(counts=count40m, MIN=MIN, MAX=MAX)
```

# 4 Session Info

```
> sessionInfo()

R version 3.0.0 (2013-04-03)
Platform: x86_64-unknown-linux-gnu (64-bit)

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=C                 LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

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

```
[8] base

other attached packages:
 [1] edgeR_3.2.3            limma_3.16.3           org.Hs.eg.db_2.9.0
 [4] RSQLite_0.11.3         DBI_0.2-6              GenomicFeatures_1.12.1
 [7] AnnotationDbi_1.22.3   Biobase_2.20.0         groHMM_0.99.0
[10] GenomicRanges_1.12.2   IRanges_1.18.0         BiocGenerics_0.6.0
[13] MASS_7.3-26

loaded via a namespace (and not attached):
 [1] biomaRt_2.16.0     Biostrings_2.28.0  bitops_1.0-5       BSgenome_1.28.0
 [5] RCurl_1.95-4.1     Rsamtools_1.12.2   rtracklayer_1.20.1 stats4_3.0.0
 [9] tools_3.0.0        XML_3.96-1.1       zlibbioc_1.6.0
```

# References


Leighton J. Core, Joshua J. Waterfall, and John T. Lis. Nascent rna sequencing reveals widespread pausing and divergent initiation at human promoters. *Science*, 322:1845–1848, 2008.

Charles G. Danko, Nasun Hah, Xin Luo, Andre L. Martins, Leighton Core, John T. Lis, Adam Siepel, and W. Lee Kraus. Signaling pathwasys differentially affect rna polymerase iii initiation, pausing and elongation rate in cells. *Molecular Cell*, 50: 212–222, 2013.

Robert C. Gentleman, Vincent J. Carey, Douglas M. Bates, Ben Bolstad, Marcel Dettling, Sandrine Dudoit, Byron Ellis, Laurent Gautier, Yongchao Ge, Jeff Gentry, Kurt Hornik, Torsten Hothorn, Wolfgang Huber, Stefano Iacus, Rafael Irizarry, Friedrich Leisch, Cheng Li, Martin Maechler, Anthony J. Rossini, Gunther Sawitzki, Colin Smith, Gordon Smyth, Luke Tierney, Jean YH Yang, and Jianhua Zhang. Bioconductor: Open software development for computational biology and bioinformatics. *Genome Biology*, 5:1845–1848, 2004.

Nasun Hah, Charles G. Danko, Leighton Core, Joshua J. Waterfall, Adam Siepel, John T. Lis, and W. Lee Kraus. A rapid, extensive, and transient transcriptional response to estrogen signaling in breast cancer cells. *Cell*, 145:622–634, 2011.

Mark D. Robinson, Davis J. McCarthy, and Gordon K. Smyth. edger: a bioconductor package for differential expression analysis of digital gene expressiondata. *Bioinformatics*, 26:139–140, 2010.