Homework 2 - Part 2

Xiyang Dai

```
library(TxDb.Dmelanogaster.UCSC.dm3.ensGene)
```

```
## Loading required package: GenomicFeatures
```

```
## Warning: package 'GenomicFeatures' was built under R version
3.0.3
```

```
## Loading required package: BiocGenerics
## Loading required package: parallel
##
## Attaching package: 'BiocGenerics'
##
## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB
##
## The following object is masked from 'package:stats':
##
##     xtabs
##
## The following objects are masked from 'package:base':
##
##     anyDuplicated, append, as.data.frame, as.vector, cbind,
##     colnames, duplicated, eval, evalq, Filter, Find, get,
##     intersect, is.unsorted, lapply, Map, mapply, match, mget,
##     order, paste, pmax, pmax.int, pmin, pmin.int, Position,
rank,
##     rbind, Reduce, rep.int, rownames, sapply, setdiff, sort,
##     table, tapply, union, unique, unlist
##
## Loading required package: IRanges
## Loading required package: GenomicRanges
## Loading required package: XVector
## Loading required package: AnnotationDbi
## Loading required package: Biobase
## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages
'citation("pkgname")'.
```

```
txdb = TxDb.Dmelanogaster.UCSC.dm3.ensGene
theGene = "FBgn0000008"

exonsByGene = exonsBy(txdb, by = "gene")
theRegions = disjoin(exonsByGene[[theGene]])

txsByGene = transcriptsBy(txdb)
theTxsIds = values(txsByGene[[theGene]])$tx_id

exonsByTranscript = exonsBy(txdb, by = "tx")
theTxs = exonsByTranscript[theTxsIds]
```

Question 1 Write code to create a matrix C with number of rows equal to the length of object theRegions and number of columns equal to the length of object theTxs, defined as follows:

```
C = matrix(nrow = length(theRegions), ncol = length(theTxs))
for (i in 1:length(theTxs)) {
    C[, i] = countOverlaps(theRegions, theTxs[i]@unlistData)
}
print(C)
```

```
##       [,1] [,2] [,3]
##  [1,]    1    0    0
##  [2,]    1    1    0
##  [3,]    0    1    0
##  [4,]    0    0    1
##  [5,]    1    1    1
##  [6,]    1    1    1
##  [7,]    1    1    1
##  [8,]    1    1    1
##  [9,]    1    1    1
## [10,]    1    1    1
## [11,]    1    1    1
## [12,]    1    1    1
## [13,]    1    1    1
## [14,]    0    1    1
```

```
library(ShortRead)
```

```
## Loading required package: Biostrings
## Loading required package: lattice
## Loading required package: Rsamtools
```

```
# the entire genomic region overlapping the disjoint gene regions
theRegion = range(theRegions)

# load reads in genomic region (extended by 100,000 bases to either
side)
theBigRegion = resize(theRegion, width = width(theRegion) + 2e+05,
fix = "center")

# there is a mismatch in chromosome names between the transcript
annotation
# we're using and the aligned read data, so let's reconcile them
theBigRegion = keepSeqlevels(theBigRegion, "chr2R")
theBigRegion = renameSeqlevels(theBigRegion, c(chr2R = "2R"))

# now load aligned reads for this region assuming the BAM file is
in the
# current directory
aln = readGAlignments("thefile_sorted.bam", param =
ScanBamParam(which = theBigRegion))

# change the chromosome names in the aligned reads object to match
the
# transcript annotation
chrNames = seqlevels(aln)
renameVector = paste("chr", chrNames, sep = "")
names(renameVector) = chrNames
aln = renameSeqlevels(aln, renameVector)

# there is a slight disagreement on the chromosome lengths between
the
# reference used to align reads and the transcript annotation.
Let's drop
# the chrU and chrUextra chromosomes
namesToKeep = seqlevels(aln)[!seqlevels(aln) %in% c("chrU",
"chrUextra")]
aln = keepSeqlevels(aln, namesToKeep)
theRegions = keepSeqlevels(theRegions, namesToKeep)

# last thing, strand is meaningless for this RNA-seq dataset, so
let's make
# all the overlap computation functions ignore it by setting strand
to '*'
strand(aln) = "*"
```

Question 2 Are all three transcripts for this gene expressed? (Answer yes/no/can't tell for each one, and why).

```
olaps = summarizeOverlaps(theRegions, aln)
counts = assay(olaps)[, 1]
print(counts)
```

```
##  [1]   0   3   4   3  17   2 156  51 277 294  32  19  24   0
```

From the counts, it is hard to tell the exact expressed isoforms. Since, the first number of extons is 0 (this is unique for the first isoform), we guess the first isoform is not expressed. The second and third are expressed.

Question 3 How is a_ij defined in this model?

In the paper, a is defined as:

$$a = lwC$$

where l is the length of exons and wis the total number of reads.

```
w = length(aln)
l = theRegions@ranges@width
A = l * w * C
print(A)
```

```
##          [,1]     [,2]     [,3]
##  [1,]    37974        0        0
##  [2,]   683532   683532        0
##  [3,]        0  3455634        0
##  [4,]        0        0 10765629
##  [5,]  4784724  4784724  4784724
##  [6,]   797454   797454   797454
##  [7,] 14999730 14999730 14999730
##  [8,]  4044231  4044231  4044231
##  [9,] 27986838 27986838 27986838
## [10,] 22936296 22936296 22936296
## [11,]  3246777  3246777  3246777
## [12,]  2240466  2240466  2240466
## [13,]  6417606  6417606  6417606
## [14,]        0   132909   132909
```

Question 4 Write down the optimization problem to solve in terms of matrix A computed above. Use θ as the parameters to estimate.

First, we need to model the likilihood function, given observed exton counts x_1, x_2, …, x_n, we want to model

$$P(x_1, x_2, \dots, x_n|\theta) = \prod_{i=1}^{n} \frac{e^{-\sum_j a_{ij}\theta_j}(-\sum_j a_{ij}\theta_j)^{x_n}}{!x_n}$$

Then, we use MLE to optimize this problem

$$\arg\max_{\theta} \log(P(x_1, x_2, \dots, x_n|\theta)) \propto \arg\max_{\theta} \log \sum_{i=1}^{n} (\sum_j a_{ij}\theta_j) - \sum_j a_{ij}\theta_j$$

Question 5 Write function loglik <- function(theta, counts, A){…} that takes a vector of parameters theta as argument and computes the log-likelihood of the count data using matrix A given current set of parameters theta.

```
loglik = function(theta, counts, A) {
    result = 0
    for (i in 1:length(counts)) {
        tmp = sum(A[i, ] * theta)
        result = result + counts[i] * log(tmp) - tmp
    }
    return(-result)
}

# loglik2 = function(theta, counts, A){ -
(t(counts)%*%log(A%*%theta) -
# sum(A%*%theta)) }
```

Question 6 Write function loglikGrad <- function(theta, counts, A) {…} that takes a vector parameters theta as argument and computes the gradient of the log-likelihood at current value of parameters theta.

```
loglikGrad = function(theta, counts, A) {
    result = theta * 0
    for (j in 1:length(theta)) {
        result_j = 0
        for (i in 1:length(counts)) {
            result_j = result_j + counts[i] * (A[i, j]/(sum(A[i, ]
* theta))) -
                A[i, j]
        }
        result[j] = result_j
    }
    return(-result)
}

# loglikGrad2 = function(theta, counts, A){
# -(t(counts)%*%(A/colSums(A*theta)) - colSums(A)) }
```

Question 7 Write a function estimateTheta <- function(counts, A) {..} that uses the loglik and loglikGrad functions to compute the Maximum Likelihood Estimate of theta.

```
estimateTheta = function(counts, A) {
    ntheta = length(A[1, ])
    # initialize \theta s.t. \sum_j \theta_j = 1
    theta = A[1, ] * 0 + 1/ntheta
    constrOptim(theta, loglik, loglikGrad, ui = matrix(c(1, 0, 0,
0, 1, 0, 0,
        0, 1), 3, 3), ci = c(0, 0, 0), counts = counts, A = A)
}

theta = estimateTheta(counts, A)
print(theta$par)
```

```
## [1] 8.513e-06 1.125e-06 2.859e-07
```

Discussion

From the optimized result we can see all three isoforms are expressed. And from the quantitive level, we can tell that the first and the second isoforms express more then the third one.