An Introduction to Sequgio

Chen Suo, Stefano Calza, Agus Salim and Yudi Pawitan

June 3, 2014

Contents

1 Introduction

This document provides a brief guide to the *Sequgio* package, which is a package for gene isoform expression and isoform-specific read distribution estimation based on RNA-seq data.

There are two components to this package. These are: i) Construct design matrices used in expression estimation. ii) Output the expression levels and optionally the read distribution, standard error of the expression estimates and etc.

The Sequgio method is motivated upon the fact that read intensity in RNA sequencing data is often not uniform, in which case standard methods would produce biased estimates. The problem is that the read intensity pattern is not identifiable from data observed in a single sample. The method accounts for non-uniform isoform-specific read distribution and gene isoform expression estimation. A statistical regularization with L_1 smoothing penalty is imposed to control the estimation. Also, for estimability reasons, the method uses information across samples from the same gene [1].

The *Sequgio* package is available at bioconductor.org and can be downloaded via `biocLite`:

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

```
> library(Sequgio)
```

For better performances the package support parallel computing via the *BiocParallel* package which is loaded automatically. For parallel processing set the parameters to the ones suiting your platform. We will use sequencial computation here.

```
> param <- SerialParam()
```

2 Example data

We demonstrate the functionality of this R package using RNA sequencing samples provided by the *RNAseqData.HNRNPC.bam.chr14*.

```
> library(RNAseqData.HNRNPC.bam.chr14)
```

2.1 Step 1: create the annotation template

The first step is to provide a `TranscriptDb` object. We will use the one provided by the package *TxDb.Hsapiens.UCSC.hg19.knou*

```
> library(TxDb.Hsapiens.UCSC.hg19.knownGene)
```

The `TranscriptDb` object must be preprocessed to generate *disjoint* regions using the `reshapeTxDb` function. We need to set the read length paramters to the one matching your experiment.

Given that the data we use are limited to chromosome 14 we subset the `TranscriptDb` to reduce computation burden.

```
> seqs <- seqnames(seqinfo(TxDb.Hsapiens.UCSC.hg19.knownGene))
> sel <- rep(FALSE,length(seqs))
> names(sel) <- seqs
> sel["chr14"] <- TRUE
> isActiveSeq(TxDb.Hsapiens.UCSC.hg19.knownGene) <- sel
```

```
> txdb <- reshapeTxDb(TxDb.Hsapiens.UCSC.hg19.knownGene,probelen = 72L,mcpar=param)
```

This step has to be repeated only if the user changes annotation database, or some paramters (with/without junctions, etc...).

2.2 Step 2: create the design matrix object

The second step is to create design matrices for each "transcriptional unit" (see references). These matrices will be used in the fitting procedure.

Several parameters can be tuned. The main one regards which kind of library is the experiment using: *paired-end* ("PE") or *single-end* ("SE", the default). This can be set with the `method` argument. The required `mulen` argument provides an estimate of the average *fragment length.*

Here we will use paired-end data.

```
> Design <- makeXmatrix(txdb,method="PE",mulen=155,sd=50,mcpar=param)
```

As for *Step 1* also this step has to be performed only once.

For demostration *Sequgio* provides the previous objects.

```
> data("TxDb")
> data("Design")
```

2.3 Step 3: import BAM files and create a *counts* matrix

Models are fit based on a matrix with read counts for every *region* in every sample. We will now import the aligned read counts in BAM files into Ras an object called 'allCounts'. To do so you need to create a `target` object storing the filenames (with full path) and sample names to be used for count matrix headings. If BAI file are available, they can be provided in the `target` object.

The resulting object (`allCounts`) will count for every exons the overlapping reads.

Let's first create the `BamFileList` object pointing to the BAM files. Data are pair-end so we set `asMates` to be TRUE.

```
> bflist <- BamFileList(RNAseqData.HNRNPC.bam.chr14_BAMFILES,asMates=TRUE)
```

Then we create a `seqCounts` object that will store some informations used in counting. A file name for file backed matrix can be provided or a random one will be generated. Also the root directory where the files are stored can be specified (defaults to working directory).

If the backing files (*.bin and *.desc) are then moved elsewhere, the files location informations in the `seqCounts` will have to be updated.

```
> seqObj <- setCounts(bflist,txdb,fileName='test')
```

We then perform the actual counting with the function `doCounts` (no return value). This function saves the counts in the shared/filebacked `big.matrix` object.

```
> doCounts(seqObj,mcpar=param)
```

We can finally import in RAM the counts creating a standard `matrix`.

```
> allCounts <- getCounts(seqObj)
```

In case the same `big.matrix` is used for more counting it should be *reset* to have counts zero.

To do so use the `resetCounts` function

```
> resetCounts(seqObj)
```

Again for ease of computation counts object is already provided.

```
> data("Counts")
```

We can see how many read we counted

```
> colSums(allCounts)
```

The counting procedure can be performed on single BAM files too or a subset of a `BamFileList`.

For a single file we can proceed as follows

```
> bfl <- BamFile(RNAseqData.HNRNPC.bam.chr14_BAMFILES[1],asMates=TRUE)
> seqObj1 <- setCounts(bfl,txdb,fileName='test')
```

and then perform the following steps as described.

For a subset of a `seqCounts` we can proceed as described in the following code, using columns indicators (*integers*)

```
> doCounts(seqObj1,mcpar=param,which.sample=c(1L,4L))
```

or similarly using sample names

```
> doCounts(seqObj1,mcpar=param,which.sample=c('ERR127306','ERR127309'))
```

This last procedure would allow to have different *nodes* of an HPC platform to count a subset of the samples i parallel (each nodes get a different `which.sample` vector), while each node can use multiple *cores* to count individual samples. All the counts will be stored in the same shared matrix.

2.3.1 Fixing QNAME

Some preprocessing pipelines deliver BAM files for paired-end reads where every mate has a slightly different QNAME. E.g. using the SRA toolkit to convert to fastq (fastq-dump), it will generate out two fastq files, and the QNAME in each of the fastq files will be appended with a ".1" for the first pairs and a ".2" for the second pairs. Similarly we might find that pairs have a different prefix.

The matching procedure implemented in *Rsamtools* requires mates to have the same QNAME, so a trimming is required. This can be achieved when declaring the `BamFile` or `BamFileList` object using the arguments `'qnamePrefixEnd'` or `'qnameSuffixStart'`.

Unique qnames aren't a problem in this sample file - just using it to demonstrate the usage.

First no trimming

```
> fl <- system.file("extdata", "ex1.bam", package="Rsamtools")
> param <- ScanBamParam(what="qname")
> bf <- BamFile(fl, asMates=TRUE)
> scanBam(bf, param=param)[[1]]$qname[1:3]
```

then trim prefix

```
> qnamePrefixEnd(bf) <- "_"
> scanBam(bf, param=param)[[1]]$qname[1:3]
```

and now trim both prefix and suffix

```
> qnameSuffixStart(bf) <- ":"
> scanBam(bf, param=param)[[1]]$qname[1:3]
```

2.4 Step 4: fit models

Using the region(exons)-by-sample counts matrix (`allCounts`) and the design matrices object (`Design`) we can now fit models.

```
> data(Counts)
> data(Design)
> iGenes <- names(Design)
> ## Fit a single 'transcriptional unit' (one element in Design)
> fit1 <- fitModels(iGenes[22],design=Design,counts=allCounts)
> # More than one using list/for loops/mclapply/etc...
> fit2 <- lapply(iGenes[21:22],fitModels,design=Design,counts=allCounts)
```

References

1]Suo C, Calza S, Salim A and Pawitan Y (2014). Joint estimation of isoform expression and isoform-specific read distribution using multisample RNA-Seq data. *Bioinformatics*, 30