Working with ChIP-Seq Data in R/Bioconductor

Suraj Menon, Tom Carroll, Shamith Samarajiwa

September 3, 2014

Contents

1 Introduction

This practical aims to introduce to analysing ChIP-seq data in R. This will include loading aligned reads and peak call data into the R enviroment, and performing various data analyses and visualisations.

R provides support for various sequencing data formats. Here we will work with aligned reads in indexed BAM files, signal coverage files in BigWig format, and BED files containing peaks called by MACS.

First we would need to load the libraries required:

```
library(chipseq)
library(GenomicRanges)
library(htSeqTools)
library(rtracklayer)
library(Rsamtools)
library(limma)
library(GenomicAlignments)
library(caTools)
```

2 Working with aligned data

2.1 Reading in data

All files needed within this library are contained within the directory '/Data For ChIP Practical/' including BAM files (.bam), BAM index files (.bam.bai), bigWig files ('bw) and Macs peaks files (.bed).

Firstly we create a *BamFileList* object for use within the R environment. Many packages assume the naming convention for BAM indicies to be '.bam.bai' although different names may be explicitly passed to these functions.

Use Case: Create a *BamFileList* object containing details of the BAM files we wish to analyse

```
dataDir <- "/Data_For_ChIP_Practical/"
bamFiles <- dir(file.path(getwd(), dataDir), pattern="*.bam$", full.name=T)
names(bamFiles) <- gsub(".bam","",basename(bamFiles))
bamFiles
bfList <- BamFileList(bamFiles)
bfList
```

Now that we have all the BAM files in a convenient list we can look at the information within the headers using `scanBamHeader()` function. This requires a path to the BAM file which we can get using the `path()` function.

Use Case: Extract the sam header for a single bam file and examine the structure of the list using the `str()` function

```
path(bfList)
samHeader <- scanBamHeader(path(bfList["TF_1"]))
str(samHeader,max.level=2)
```

We get a list containing two components: The first is 'targets' which contains a list of chromosomes used in the alignment. The second is 'text' which contains information including the species and alignment method used.

Use Case: Explore the information provided in the sam header. Find how the data has been sorted, the aligner used and what species the data has been aligned to.

```
samHeader[[1]]$targets
samHeader[[1]]$text
samHeader[[1]]$text["@HD"]
samHeader[[1]]$text["@PG"]
samHeader[[1]]$text["@CO"]
```

BAM files usually contain a lot of information and it is often more feasible to deal with a single chromosome at a time for processing and analysis.

Here we will set up parameters in order to select only reads aligning to Chromosome 1

Use Case: Extract information about chromosome 1 coordinates to set up filtering parameters using the `ScanBamParam()` function.

```
chr1dat <- samHeader[[1]]$targets["chr1"]
chr1dat
chr1range <- GRanges(seqnames=names(chr1dat),ranges=IRanges(1,chr1dat))
param <- ScanBamParam(which=chr1range)
param
```

Note that in this example, we use the default `ScanBamParam` 'flag' argument. However we could use this to build further filters (e.g. to remove duplicates or other potential artifacts) using the `scanBamFlag()` function. Look at the options for this function for more details using `?scanBamFlag`

Use Case: Having selected an area of interest, read in read data from a single BAM file using the parameters set using the `readGAlignments()` function. Convert this to a *GenomicRanges* object and inspect the data contained within. Calculate the average read length of the selected reads (in case some reads were trimmed).

```
alignDat <-  readGAlignments(path(bfList["TF_1"]),param=param)
alignGR <- granges(alignDat)
seqlevels(alignGR) <- "chr1"
median(width(alignGR))
```

2.2 Coverage and fragment lengths

We can use the aligned reads to calculate genomic coverage, i.e. the profile of sequencing depth along a genome. The `coverage()` function allows creation of coverage profiles in the Run Length Encoded Lists (*RLEList*) format which comprises of compressed vectors of read depths at all posisions in a chromosome with different chromosomes as different list elements. This format allows for long stretches of genomic locations at the same depths to be compressed.

Use Case: Separate reads aligning to the positive and negative strands using the `strand()` accessor function. Examine the resulting object to check for separation. Use the `coverage()` function to create coverage vectors.

```
strand(alignGR)
alignPos <- alignGR[strand(alignGR) == "+"]
alignNeg <- alignGR[strand(alignGR) == "-"]
strand(alignPos)
strand(alignNeg)
posCov <- coverage(alignPos)
negCov <- coverage(alignNeg)
posCov
negCov
```

We will now focus on a specific region of known binding (chr1:211646604-211649812). We can visualise

the coverage around this region for each strand. The distance between the positions where the positive and negative strands show maximum coverage can give an indication of how much the reads aligning to the two strands are shifted by.

Use Case: Extract coverage values for the region of interest. Use the `runmean()` function from the caTools package to derive smoothed coverage values for plotting. Plot these coverage values for the positive and negative strands. Calculate the approximate strand shift.

```
coords <- c(211646604:211649812)
subPosCov <- posCov$chr1[coords]
subNegCov <- negCov$chr1[coords]
smoothPosCov <- caTools::runmean(as.vector(subPosCov),20)
smoothNegCov <- caTools::runmean(as.vector(subNegCov),20)

pdf("NoReadExtension.pdf", width=14)
plot(smoothNegCov, col="red", type="l",
      ylab="Smoothed Read Depth 20bp Window",
      xlab="bp from range start")
lines(smoothPosCov,col="blue",type="l")
abline(v=which.max(smoothNegCov), col="red", lty=2)
abline(v=which.max(smoothPosCov), col="blue", lty=2)
dev.off()
which.max(smoothNegCov) - which.max(smoothPosCov)
```

Estimating fragment length is an important step in ChIP-Seq data processing as a QC measure and as a precursor to further analyses such as peak calling to identify transription factor binding sites. Here we estimate fragment length using the cross-coverage methodology. This involves shifting reads from one strand and attempting the find the shift required such that the proportion of the genome covered by the reads is minimized.

Use Case: Estimate fragment length by cross-coverage method using the `estimate.mean.fraglen()` function from the *chipseq* package [NOTE using the median function is redundant since we are working with a single chromosome in this example]. Extend reads to estimated fragment length. Plot coverage for extended reads aligned to positive and negative strands.

```
fragLen <- median(estimate.mean.fraglen(c(alignPos ,alignNeg),method="coverage"))
fragLen
extendReads <- resize(alignGR, fragLen, fix="start")
median(width(alignGR))
median(width(extendReads))
extPos <- extendReads[strand(extendReads) == "+"]
extNeg <- extendReads[strand(extendReads) == "-"]
smoothPosCovExt <- caTools::runmean(as.vector(coverage(extPos)$chr1[coords]),20)
smoothNegCovExt <- caTools::runmean(as.vector(coverage(extNeg)$chr1[coords]),20)
pdf("Cross_Coverage_ReadExtension.pdf")
plot(smoothNegCovExt,col="red",type="l",
      ylab="Smoothed Read Depth 20bp Window",
```

```
      xlab="bp from range start")
lines(smoothPosCovExt, col="blue",type="l")
dev.off()
which.max(smoothNegCovExt) - which.max(smoothPosCovExt)
```

2.3 Dispersion of coverage and duplication

A hallmark of good ChIP data is the inequality of coverage that occurs when there is enrichment of reads to certain portions of the genome (e.g. transcription factor binding sites). Two such metrics that attempt to summarise the distribution of signal dpeths across the genome are the Standardised Standard Deviation (SSD) and Gini scores of coverage. Functions to calculate these are provided in the *htSeqTools* package.

Use Case: Calculate SSD and Gini scores for the aligned data using the `ssdCoverage()` and `giniCoverage()` functions respectively.

```
ssdCoverage(alignGR)
giniCoverage(alignGR)
```

Calculate the same for other samples. How do these values compare between ChIP samples and input samples?

htSeqTools also provides functions that allow examination of the number of duplicate reads in a dataset. Using statistical modelling, *htSeqTools* also identifies signficantly over-duplicated reads (which are likely to be artifacts) and filters them.

Use Case: Use the `tabDuplReads()` function to examine the duplication rates. Then use the `filterDuplReads()` function to remove significantly over-duplicated regions.

```
numDups <- tabDuplReads(RangedData(alignGR))
numDups
dupFilt <- filterDuplReads(RangedData(alignGR))
tabDuplReads(dupFilt)
```

3 Working with peak data

3.1 Reading in peaksets

In this section, we will introduce working with the peaks called in ChIP-Seq data within R. We will perform various steps for processing and visualising the data, as well as further downstream analyses that allow for exploration of this data in biological contexts.

The peaks used in this practical were generated using the popular peak caller MACS. MACS attempts to identify genomic windows which are enriched for sample reads when compared to the input as well

as the global expected rate of reads in that window.

We will start by examining the BED files generated by MACS containing information about peaks called.

Use Case: Read in a MACS peak BED file using the standard R `read.delim()` function. Examine the data contained within

```
macsBed <- read.delim(file.path(getwd(), dataDir, "TF_1_peaks.bed"),
                sep="\t",header=F)
head(macsBed)
```

The resulting data frame is one where each row represents a peak. THe columns represent the chromosome, start and end coordinates of the peak, peak name and MACS peak score.

We will now convert this into a *GRanges* object that will allow for ease of further data processing and analysis. The chromosome, strand, start and end positions are all that is needed for a GRanges object; all other information is stored in the elementMetadata slot. Since we have no strand information we will set this to '*' to mark it as unknown.

Use Case: Convert the peak data into a *GRanges* object. Query the *GRanges* class to understand the various parameters and accessor functions. Find out how many peaks there are in the file and the widths of these peaks.

```
macsGR <- GRanges(seqnames=as.vector(macsBed[,1]),
          IRanges(start=as.numeric(as.vector(macsBed[,2])),
                  end=as.numeric(as.vector(macsBed[,3]))),
          strand=rep("*", nrow(macsBed)))
elementMetadata(macsGR) <- macsBed[,-c(1:3)]
colnames(elementMetadata(macsGR)) <- c("Peak_ID","Score")
macsGR
length(macsGR)
width(macsGR)
```

3.2 Overlapping, merging and filtering peaksets

For the sake of convinience we will use three pre-written functions (`Bed2GRanges`, `MakeConsensusSet`) and `GetAverageSignalOverRanges()` that are provided in a (.R) file along with the raw data. This file can be read in using the `source()` function.

Use Case: Read in the R script in the practical data directory using the `source()` function. Use the (`Bed2GRanges` function to read in all the BED files in the directory. Find the number of peaks overlapping between two peaksets.

```
source(file.path(getwd(), dataDir, "ChIPSeq_functions.R"))

peakFiles <-  dir(file.path(getwd(),"/Data_For_ChIP_Practical/"),
                  pattern="*_peaks.bed",full.names=T)
peakFiles
```

```
pfList <- lapply(peakFiles,Bed2GRanges)
names(pfList) <- gsub("_peaks.bed","",basename(peakFiles))

## Peaks in file 1 which overlap peaks in file 2
pfList[[1]][pfList[[1]] %over% pfList[[2]]]
## Peaks in file 2 which overlap peaks in file 1
pfList[[2]][pfList[[2]] %over% pfList[[1]]]
## Peaks unique to file 2
pfList[[2]][!pfList[[2]] %over% pfList[[1]]]
```

A common task ChIP-seq analyses is to identify peaks which occur across multiple peaksets, for example, across replicates of a group of samples. One approach is to construct a consensus set of peaks by "reducing" all peaks to a common, non-overlapping set and then using this consensus set as the basis for further analyses.

Use Case: Use the `MakeConsensusSet()` from the sourced R script that was read in to create a consensus peak set by merging all the peaksets in the directory.

```
mergedPeaks <- MakeConsensusSet(pfList)

## Peaks in file 1 which overlap peaks in consensus set
pfList[[1]][pfList[[1]] %over% mergedPeaks]
## Peaks in file 2 which overlap peaks in consensus set
pfList[[2]][pfList[[2]] %over% mergedPeaks]
## Peaks unique to file 2
pfList[[2]][!pfList[[2]] %over% mergedPeaks]

## filter to only keep peaks in chr 1
mergedPeaks <- mergedPeaks[seqnames(mergedPeaks) %in% "chr1"]
seqlevels(mergedPeaks) <- "chr1"
```

Note: In this practical we are filtering the merged peaks for only those on chromosome 1 in order to make the object compatible with the truncated BAM and bigWig files that only contain data for chromosome 1

The object containing the merged peaks also contains information on the occurrence of peaks used to construct the consensus set. There are columns representing each peak file comprised of 0s and 1s representing whether a consensus peak occured in that sample (1) or not (0).

Use Case: Use the metadata in the consensus peakset object to create a Venn diagram representing peak co-occurence across the different peaksets.

```
## Extract co-occurence information from mergedPeaks object
occ <- vennCounts(as.data.frame(elementMetadata(mergedPeaks)))
occ

pdf("Peak_Overlap_VennDiagram.pdf")
```

```
vennDiagram(occ)
dev.off()
```

Use Case: Use the metadata in the consensus peakset object to extract peaks that (a) occur in both TFs but not in the CoTF peakset and (b) occur in all three peaksets. How many peaks are there in each case? Do they match up with what we see in the Venn diagram?

```
peaksTFonly <- mergedPeaks[elementMetadata(mergedPeaks)$TF_1 ==1
                  & elementMetadata(mergedPeaks)$TF_2 ==1
                  & elementMetadata(mergedPeaks)$CoTF ==0]
peaksTFandCoTF <- mergedPeaks[elementMetadata(mergedPeaks)$TF_1 ==1
                     & elementMetadata(mergedPeaks)$TF_2 ==1
                     & elementMetadata(mergedPeaks)$CoTF ==1]
length(peaksTFonly)
length(peaksTFandCoTF)
```

3.3 Coverage and read counts for peaks

Having refined our regions of interest we can now visualise the coverage of reads at these locations. We can extract average read coverage of a peakser from the bigwig files for these data using the `GetAverageSignalOverRanges()` function that we sourced earlier.

Use Case: Use the `GetAverageSignalOverRanges()` function to extract average read coverage from the bigWig files for TF1 and CoTF for the refined peaksets using an 1000BP window around the peak centres. Plot these data to see if these coverages differ between the refined peaksets.

```
bwCoTF <- file.path(getwd(), dataDir, "CoTF.bw")
bwTF1 <- file.path(getwd(), dataDir, "/TF_1.bw")
bwTF2 <- file.path(getwd(), dataDir, "/TF_2.bw")

covCommon_TF1 <- GetAverageSignalOverRanges(bwTF1, peaksTFandCoTF, 1000)
covCommon_TF2 <- GetAverageSignalOverRanges(bwTF2, peaksTFandCoTF, 1000)
covCommon_CoTF <- GetAverageSignalOverRanges(bwCoTF, peaksTFandCoTF, 1000)
covTFOnly_TF1 <- GetAverageSignalOverRanges(bwTF1, peaksTFonly, 1000)
covTFOnly_TF2 <- GetAverageSignalOverRanges(bwTF1, peaksTFonly, 1000)
covTFOnly_CoTF <- GetAverageSignalOverRanges(bwCoTF, peaksTFonly, 1000)

yMax <- max(covCommon_TF1, covCommon_CoTF, covTFOnly_TF1, covTFOnly_CoTF)

pdf("PeakCoverage.pdf", width=10)
par(mfrow=c(1,2))
plot(c(-499:500), covCommon_TF1, col="red", ylim=c(0, yMax), type="l",
      main="Common peaks", ylab="bigWig coverage",
      xlab="Distance from peak centre")
lines(c(-499:500), covCommon_TF2, col="green")
```

```
lines(c(-499:500), covCommon_CoTF, col="blue")
abline(v=0, lty=2)
legend("topright", fill=c("red", "green", "blue"), legend=c("TF1", "TF2", "CoTF"))
plot(c(-499:500), covTFOnly_TF1, col="red", ylim=c(0, yMax), type="l",
      main="TF only peaks", ylab="bigWig coverage",
      xlab="Distance from peak centre")
lines(c(-499:500), covTFOnly_TF2, col="green")
lines(c(-499:500), covTFOnly_CoTF, col="blue")
legend("topright", fill=c("red", "green", "blue"), legend=c("TF1", "TF2", "CoTF"))
abline(v=0, lty=2)
dev.off()
```

We can query BAM files in the context of defined regions, e.g. the consensus peakset. The `summarizeOverlaps()` function allows us to calculate read counts for specified genomic regions.

Use Case: Use the `summarizeOverlaps()` function to find get read counts for the consensus peakset using the BAM files. Add this count information to the metadata of the consensus peakset object. Do the counts agree with the peak co-occurence information?

```
readCounts <- summarizeOverlaps(mergedPeaks, bfList)
head(assays(readCounts)$counts)

elementMetadata(mergedPeaks) <- cbind(as.data.frame(
   elementMetadata(mergedPeaks)), assays(readCounts)$counts)
head(mergedPeaks)
```

3.4 Annotation and biological exploration of peaks

The *ChIPpeakAnno* package allows for investigation of peaks in a biological context. It provides batch biological annotation of peaks and includes functions for various tasks such as retrieval of sequences around peaks, assessment of enrichment of Gene Ontology (GO) terms, finding the nearest gene/exon/miRNA/custom feature, etc. In this section we will use some of these functions

Use Case: Load the *ChIPpeakAnno* package and use the `annotatePeakInBatch` to annotate a peakset. Write out the annotated data into a file and examine it.

```
library(ChIPpeakAnno)
##convert GRanges to RangedData format
peakSetRD = as(peaksTFandCoTF, "RangedData")
## Load gene locations for human genome hg19
data(TSS.human.GRCh37)

annotatedPeaks = annotatePeakInBatch(peakSetRD, AnnotationData = TSS.human.GRCh37)
write.table(as.data.frame(annotatedPeaks), file = "annotatedPeakList.xls",
              sep = "\t", row.names = FALSE)
```

Use Case: Use the information in the annotated peaks to the density distribution of distances of peaks to nearest gene transcription start sites.

```
nearestTSSdist = annotatedPeaks$distancetoFeature[
                 !is.na(annotatedPeaks$distancetoFeature) &
                 annotatedPeaks$fromOverlappingOrNearest == "NearestStart"]

pdf("DistanceToNearestTSS.pdf")
plot(density(nearestTSSdist), xlab = "Distance To Nearest TSS")
dev.off()
```

Use Case: Use the `annotatePeakInBatch` to carry out GO term enrichment analysis on the annotated peakset. For this the *ChIPpeakAnno* requires annotation contained in the *org.Hs.eg.db* package which will need to be loaded. Extract GO Biological Process terms results and examine these. Which terms appear to be most enriched?

```
library(org.Hs.eg.db)
goAnalysis = getEnrichedGO(annotatedPeaks, orgAnn = "org.Hs.eg.db", maxP =0:01,
                 multiAdj = TRUE, minGOterm = 10, multiAdjMethod = "BH" )
names(goAnalysis)
bpResult <- goAnalysis$bp
bpResult <- bpResult[order(bpResult$pvalue, decreasing=F),]
```

Use Case: Get the coordinates of windows of 150BP around the centres of peaks in a peakset. Use the `getAllPeakSequence` to retieve the genomic sequences at these coordinates. For this the *ChIPpeakAnno* requires annotation contained in the *BSgenome.Hsapiens.UCSC.hg19* package which will need to be loaded. Write the sequences out in Fasta format.

```
library(BSgenome.Hsapiens.UCSC.hg19)

peakCentres <- resize(peaksTFandCoTF, fix="center", 150)
peakCentresRD = as(peakCentres,"RangedData")
peakSeq <- getAllPeakSequence(peakCentresRD, upstream=0, downstream=0,
              genome = Hsapiens)
sequences <- peakSeq$sequence
names(sequences) <- paste(peakSeq$space, start(peakSeq), end(peakSeq), sep="_")
seqXstring <- DNAStringSet(sequences)
writeXStringSet(seqXstring,file="sequences.fasta",width=150)
```

Go to the Meme-ChIP suite website and upload the Fasta file to identify the motif of our co-occurring TF within the peak set. http://meme.nbcr.net/meme/cgi-bin/meme-chip.cgi