chip-seq data analysis
2011 BioConductor LabSession: “A Bioconductor pipeline for the analyis of ChIP-Seq experiments.”

Xuekui Zhang, Eloi Mercier and Arnaud Droit

July 22, 2011

1 Introduction

This package *ChipSeq* provides all necessary data for the chip-seq sections of the 2011 BioConducor Labsession. This vignette also includes all commands that we will be used throughout the lab sessions(? ?).

Part I
chip-seq

2 Data Input

For your convenience, the experimental data required in this package have already been pre-formatted and can simply be loaded with the following commands:

```
> library(ChipSeq)
> # Chip-seq ER data
> # This will load two datasets
> data(ER)
> # ChIP-Seq FOXA1 data
> data(FOXA1)
> # Mappability profiles
> data(mapp)
```

then the ER chip-seq data(?) will be loaded in your workspace and you can quickly have a look by typing

```
> ER.E2
```

```
A GenomeData instance
chromosomes(2): chr21 chr22

> ER.ethl

A GenomeData instance
chromosomes(2): chr21 chr22
```

We have also included the raw data in the package, and the following commands should get you the pre-formatted data

```
> library(PICS)
> library(ShortRead)
> # ER data (Eland file)
> # Get the path of the data
> path <- system.file("extdata/chip-seq/ER",package = "ChipSeq")
> # Grep the treatment file
> E2.file<-list.files(path, pattern = "E2",full.names = FALSE)
> # Grep the control file
> ethl.file<-list.files(path, pattern = "ethl",full.names = FALSE)
> # Set some filters
> filtChr<- chromosomeFilter("chr")
> filtUnique<-occurrenceFilter(min=1L, max=1L)
> filtStrand<-strandFilter(strandLevels=c("+","-"))
> filtOverall<-compose(filtChr,filtUnique,filtStrand)
> # Now we use ShortRead to read the data
> ER.E2<-readAligned(path,type="SolexaExport",pattern=E2.file,filter=filtOverall)
> ER.E2<- as(ER.E2, "GRanges")
> ER.E2<-as(ER.E2,"RangedData")
> ER.E2<-as(ER.E2,"GenomeData")
> # Fix the name
> names(ER.E2)<-sub(".fa","",names(ER.E2))
> names(ER.E2)<-sub("hs_ref_","",names(ER.E2))
> # Only keep chromosome 21 and 22
> ER.E2<-ER.E2[c("chr21","chr22")]
> #Do the same for the IP file
> ER.ethl<-readAligned(path,type="SolexaExport",pattern=ethl.file,filter=filtOverall)
> ER.ethl<- as(ER.ethl, "GRanges")
> ER.ethl<-as(ER.ethl,"RangedData")
> ER.ethl<-as(ER.ethl,"GenomeData")
> # Fix the name
> names(ER.ethl)<-sub(".fa","",names(ER.ethl))
> names(ER.ethl)<-sub("hs_ref_","",names(ER.ethl))
> # Only keep chromosome 21 and 22
> ER.ethl<-ER.ethl[c("chr21","chr22")]
> # save(ER.ethl,ER.E2,file="ER.rda")
```

Exercise 2.1 *Do the same with FOXA1 chip-seq data.*

Here our aligned reads are stored in *GenomeData* objects whereas our mappability profiles are stored in *RangedData* objects. Note that the mappability profile is read length dependent. For each chromosome, a mappability profile for a specific read length (e.g. 36 bp) consists of a vector that lists an estimated read mappability 'score' for each base pair in the chromosome. A score of one at a genomic position means that we should be able to uniquely align a read that overlaps that position, while a score of zero indicates that no read of that length should be uniquely alignable at that position. As noted above, typically only reads that map to unique genomic locations are retained for analysis. For convenience, and because transitions between mappable and non-mappable regions are often much shorter than the regions, we compactly summarize each chromosome's mappability profile as a disjoint union of non-mappable intervals that specify only zero-valued profile regions

The *GenomeData* object *GenomeData* formally represents genomic data as a list, with on element per chromosome in the genome.

This class facilitates storing data on the genome by formalizing a set of metadata fields for storing the organism (e.g. Mmusculus), genome build provider (e.g. UCSC), and genome build version (e.g. mm9).

The data is represented as a list, with one element per chromosome (or really any sequence, like a gene). There are no constraints as to the data type of the elements.

```
> # This loads two datasets FOXA1.IP and FOXA1.INP
> data(FOXA1)
> FOXA1.IP

A GenomeData instance
chromosomes(2): chr21 chr22

> names(FOXA1.IP)

[1] "chr21" "chr22"

> FOXA1.IP["chr21"]

A GenomeData instance
chromosomes(1): chr21

> names(FOXA1.IP[[1]])
```

```
[1] "-" "+"

> names(FOXA1.IP[[2]])

[1] "-" "+"

> metadata(FOXA1.IP)

$organism
NULL

$provider
NULL

$providerVersion
NULL

> metadata(FOXA1.IP)$organism<-"FOXA1 in MCF7 on Hsapiens"
> metadata(FOXA1.IP)$provider<-"UCSC"
> metadata(FOXA1.IP)$providerVersion<-"hg18"
```

Exercise 2.2 *Do the same with the ER chip-seq data.*

3 Statistical Analysis

3.1 Genome segmentation

Because ChIP-seq aligned-read data are usually sparse, consisting largely of regions in which few or no reads are observed, we first preprocess the read data by segmenting the genome into regions, each of which has a minimum number of reads that aligned to forward and reverse strands. Using the data we have read and formatted, as described above, we now use the *segmentReads* function, as follows,

```
> data(ER)
> data(mapp)
> seg<-segmentReads(data=ER.E2, dataC=ER.ethl, mapp36)
> summary(seg)

** Experiment information **
Chromosomes interogated: chr21 chr22
Number of reads in IP:  41909  and in control:  62025
** Segmentation parameters **
The following settings were used:
  Sliding window half width:  250
  Step size:  20
  Minimum number of reads:  2
** Segmentation summary **
Number of segmented regions: 312
Summary on the number of Forward/Reverse reads per region:
  Forward:
     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   3.00    4.00    8.00   13.07   15.00  165.00
  Reverse:
     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   3.00    4.00    8.00   13.21   15.00  199.00
Summary on the number of control Forward/Reverse reads per region:
  Forward:
     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.000   0.000   0.000   1.205   1.000  18.000
  Reverse:
     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.000   0.000   0.000   1.295   1.000  34.000
** Mappability summary **
Non mappable intervals cover an average  0.1111147 % of all regions
```

the returned value is a *segReadsList* object. Each element of the *segReadsList* contains the reads for the corresponding 'candidate' region as well as the mappability intervals intersecting the region.

3.2 Data smoothing and PICS processing:

Now that we have created our seg object, we are ready to fit *PICS* to each region, this is automatically done with the *PICS* function.

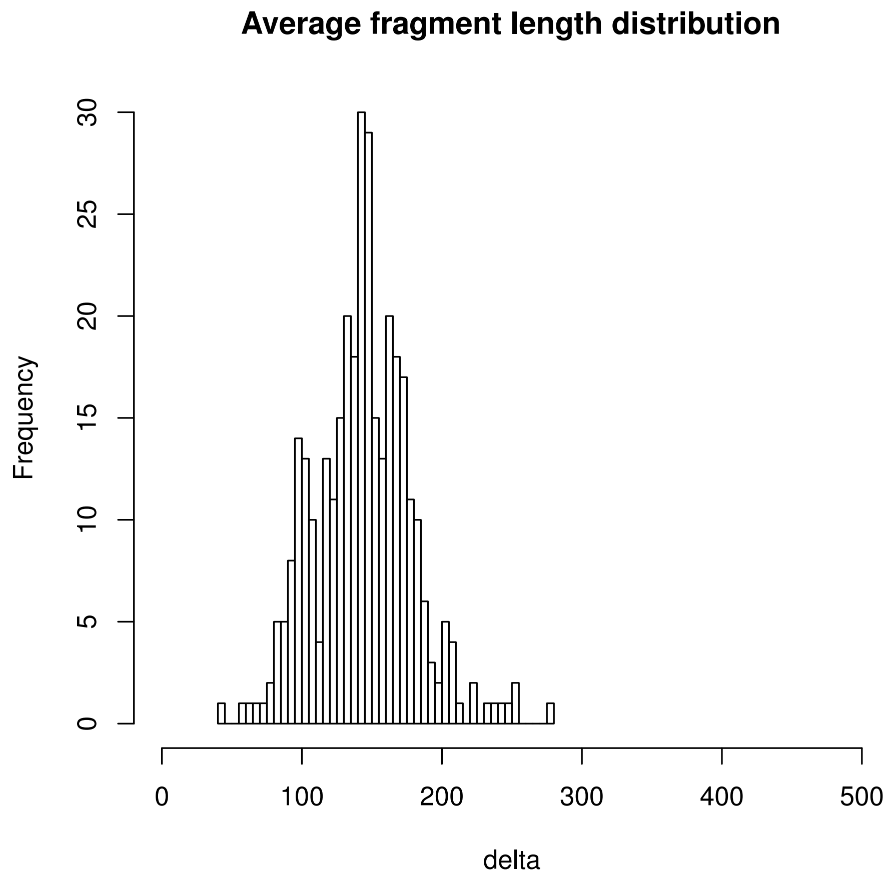
```
> # This might take about a few minutes on a single cpu
> pics<-PICS(seg,dataType="TF")
```

The *picsList* object and accessors: The object returned by the *PICS* function is an S4 class containing all necessary information (e.g. parameters, scores, etc). We have implemented numerous accessors for you to efficiently retrieve important information from such an object. All of them are documented in the *PICS* vignette, available with the package, but we review a few important accessors here:

```
> #Get the location of the binding sites (mid-point of the motifs).
> mu<-mu(pics)
> # Get the fragment length estimates from all binding events
> delta<-delta(pics)
> summary(delta)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  42.76  122.10  144.40  144.10  165.20  278.90

> # Get the enrichment score from all binding events
> score<-score(pics)
> summary(score)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.592   4.440   7.400  12.450  14.060  98.270
```

These are simple vectors containing all estimated parameters across candidate regions, and one can see that the average fragment size distributions is about 179 bps. In order to better visualize the fragment length estimates across binding events, we can use a simple histogram.

```
> # For clarity, we focus on (0,500)
> hist(delta,xlim=c(0,500),50,main="Average fragment length distribution")
```



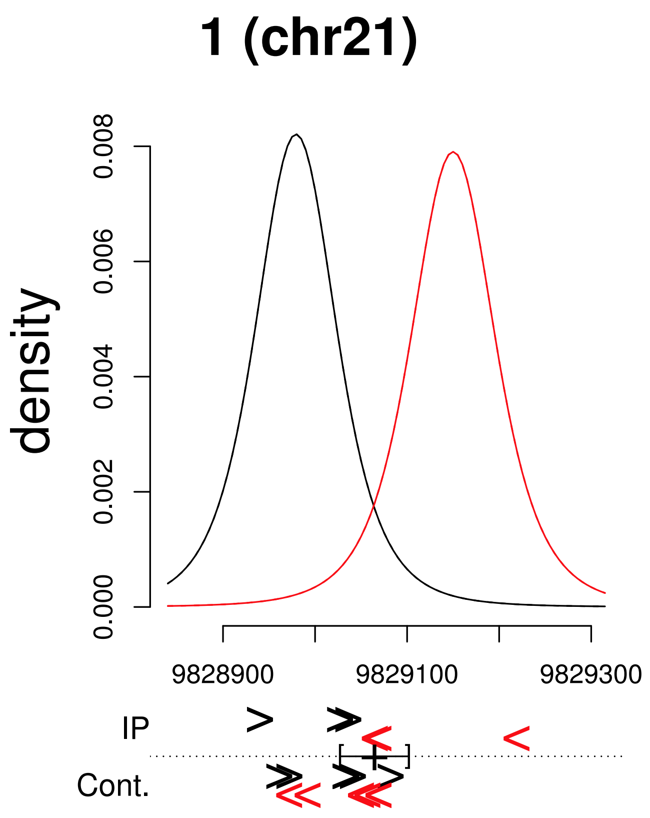
In PICS, each candidate region (a list element of seg) can contain multiple binding sites, and to summarize the number of events/region we can use the K accessor which will give us the number of binding events estimated for each region.

```
> # Retrieve the number of events per region
> nEvents<-K(pics)
> # Tabulate these numbers
> table(nEvents)
```

```
nEvents
  1   2
288  24
```

We have also included a simple plotting method for visualizing a candidate region with the PICS estimated parameters, as follows,

```
> plot(pics[2],seg[2])
```



3.3 Detecting enriched regions:

The next step would consist of selecting a list of estimated binding events to be prioritized for further analysis (e.g. motif analysis, or correlation with annotations). In the absence of control data, this needs to be done arbitrarily. For example, one could want to focus on all regions that have an enrichment score greater than 4. This can simply be done when exporting our "pics" object to a *RangedData* object, as follows, with the appropriate filter

```
> # Filter atypical peaks
> myFilter<-list(score=c(4,Inf),delta=c(50,300),se=c(0,50),
+ sigmaSqF=c(0,22500),sigmaSqR=c(0,22500))
> # Make a RangedData Object
> RD<-makeRangedDataOutput(pics, type="bed", filter=myFilter)
```

where we used the type bed for export, other export types (e.g. wig, fixed, etc) are also available. Please refer to the PICS vignette and man pages for more details. Note that above, we only keep the binding events that have a score greater than 4, a delta value (average fragment length) between 50 and 300, and standard deviation (strand specific peak width) less than 150 ($150^2 = 22500$). If one wishes to export all events with typical filters (and score>1), we provide a shortcut with the *as* method as follows:

```
> RD<-as(pics,"RangedData")
```

similarly if one wishes to export all events as a *data.frame* with all PICS parameters without filtering, we can use

```
> DF<-as(pics,"data.frame")
```

The *makeRangedDataOutput* can also be used to produce a 'wig' type track with base level scores, as follows,

```
> # Filter atypical peaks
> myFilter<-list(score=c(1,Inf),delta=c(50,300),se=c(0,50),sigmaSqF=c(0,22500)
+ ,sigmaSqR=c(0,22500))
> # Make a RangedData Object
> RDwig<-makeRangedDataOutput(pics, type="wig", filter=myFilter)
```

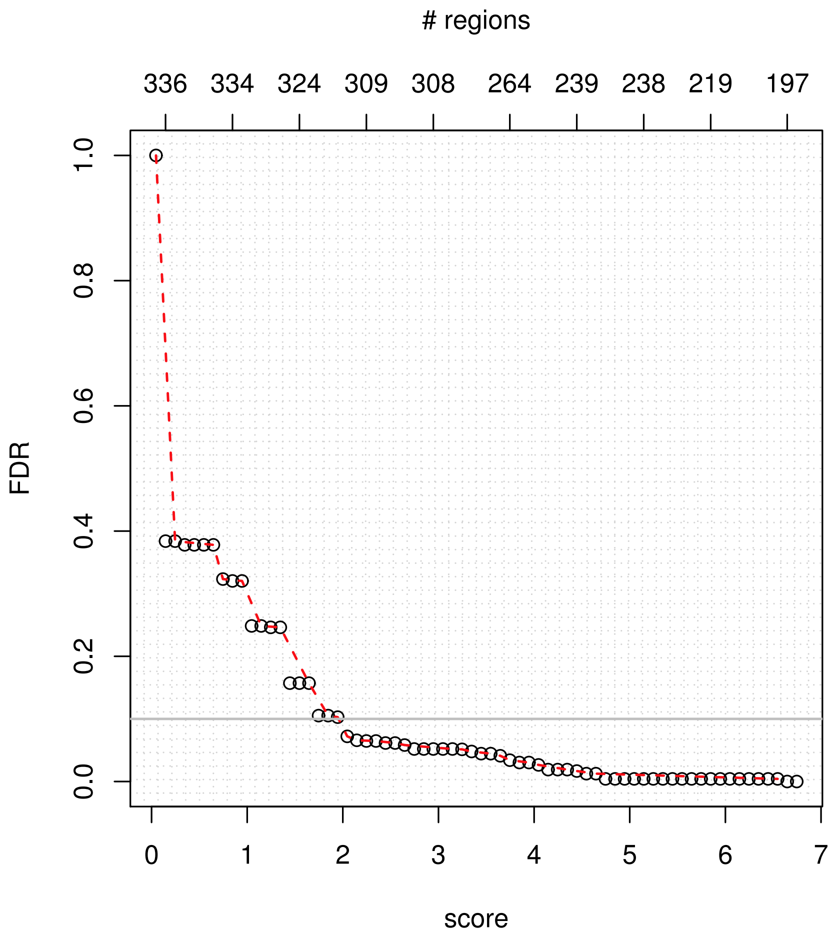
3.4 FDR calculation

In the presence of the control, it is possible to generate an FDR curve that can be used to select an appropriate threshold. The first step is to rerun the same analysis after swapping the control and IP samples, as follows:

```
> segC<-segmentReads(data=ER.ethl, dataC=ER.E2, mapp36)
> picsC<-PICS(segC,dataType="TF")
```

note that we have created a method for plot. If you input two pics objects, an FDR curve will be generated. **Note that the second argument needs to be the control.**

```
> plot(pics,picsC)
```



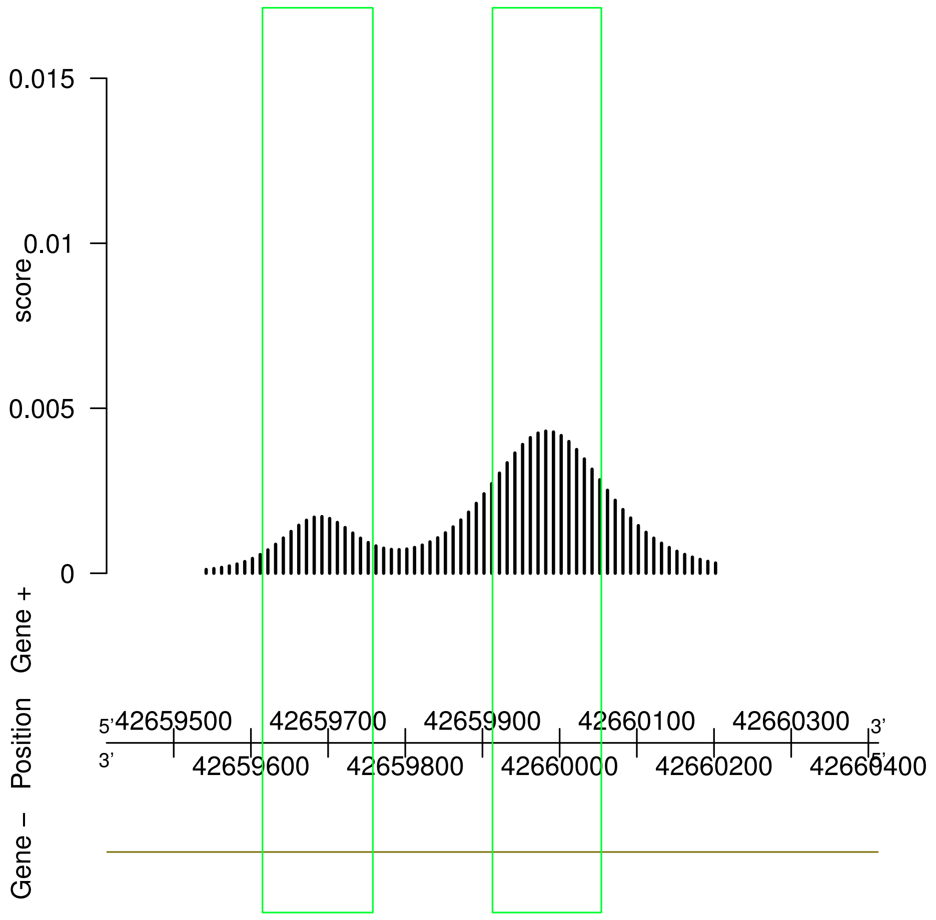
and one can see that a threshold score of 2 would lead to an estimated FDR of about 10% with about 300 regions If we want to use a score of 2 as threshold and filter atypical regions we can simply use

```
> # Filter atypical peaks
> myFilter<-list(score=c(2,Inf),delta=c(50,300),se=c(0,50),
+ sigmaSqF=c(0,22500),sigmaSqR=c(0,22500))
> # Make a RangedData Object
> RD<-makeRangedDataOutput(pics, type="bed", filter=myFilter)
```

4 Visualization

The visualization of enriched regions and track lines is possible thanks to `GenomeGraphs` and `rtracklayer`. We first explore the `GenomeGraphs` package. In the example below, we will graph our enriched regions in a subset of chromosome 21 along with the nearest genes,

```
> library(GenomeGraphs)
> # Here I use the current genome build
> mart = useMart("ensembl", dataset = "hsapiens_gene_ensembl")
> genomeAxis<-makeGenomeAxis(add53 = TRUE, add35 = TRUE)
> RDwig1<-RDwig["chr21"]
> RD1<-RD["chr21"]
> minbase<-start(RD1[2,])-500
> maxbase<-start(RD1[2,])+500
> genesplus<-makeGeneRegion(start = minbase, end = maxbase, strand = "+",
+  chromosome = 21, biomart = mart)
> genesmin<-makeGeneRegion(start = minbase, end = maxbase, strand = "-",
+ chromosome = 21, biomart = mart)
> score = makeBaseTrack(value=score(RDwig1), base = start(RDwig1),
+ dp = DisplayPars(lwd=2,color="black", type="h"))
> rectList<- makeRectangleOverlay(start = start(RD1), end = end(RD1),
+ region = c(1, 4), dp = DisplayPars(color = "green", alpha = 0.1))
> gdPlot(list("score" = score, "Gene +" = genesplus, Position = genomeAxis,
+ "Gene -" = genesmin), minBase = minbase, maxBase = maxbase, labelCex = 1,
+ overlays=rectList)
```



Now we look at the `rtracklayer` package. `rtracklayer` is basically allowing us to interact with the UCSC genome browser directly from R.

```
> library(rtracklayer)
> ## start a session
> RD_GR<-as(RD,"GRanges")
> session <- browserSession("UCSC")
> track(session, "targets") <- RD_GR
> subtrack<-RD_GR[1]
> view <- browserView(session, subtrack * -10, pack = c("Conservation","RepeatMasker"
```

Of course, you always have the option to export the results as wig/bed and load it in your favorite browser. Export functionalities are also provided by the `rtracklayer` package.

```
> library(rtracklayer)
> # These functions are provided by rtracklayer
> export(RD,"ERregions.bed")
> export(RDwig,"ERscore.wig")
```

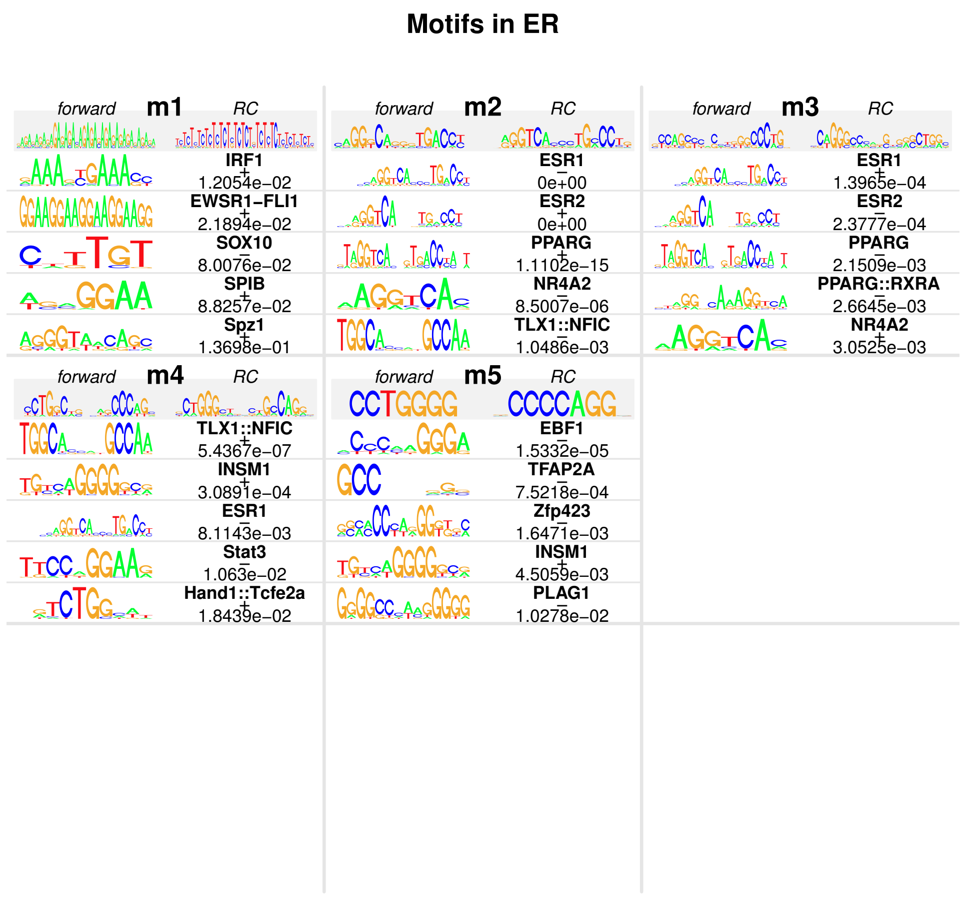
5 De Novo motif discovery:

After having detected enriched regions, it is common to perform sequence analysis to detect biologically relevant motifs that can be used to validate our regions and/or to gain novel insights about the biology of gene regulation. In collaboration with Leiping Li, we have developed an R package specifically designed to work on large set of sequences typically return by a ChIP-Seq experiment. `rGADEM` combines spaced dyads and an expectation-maximization (EM) algorithm. Candidate words (four to six nucleotides) for constructing spaced dyads are prioritized by their degree of overrepresentation in the input sequence data. Spaced dyads are converted into starting position weight matrices (PWMs). `rGADEM` then employs a genetic algorithm (GA), with an embedded EM algorithm to improve starting PWMs, to guide the evolution of a population of spaced dyads toward one whose entropy scores are more statistically significant. Spaced dyads whose entropy scores reach a pre specified significance threshold are declared motifs. Here we will perform a motif analysis of the set of ER enriched regions returned by PICS using an FDR of 10%

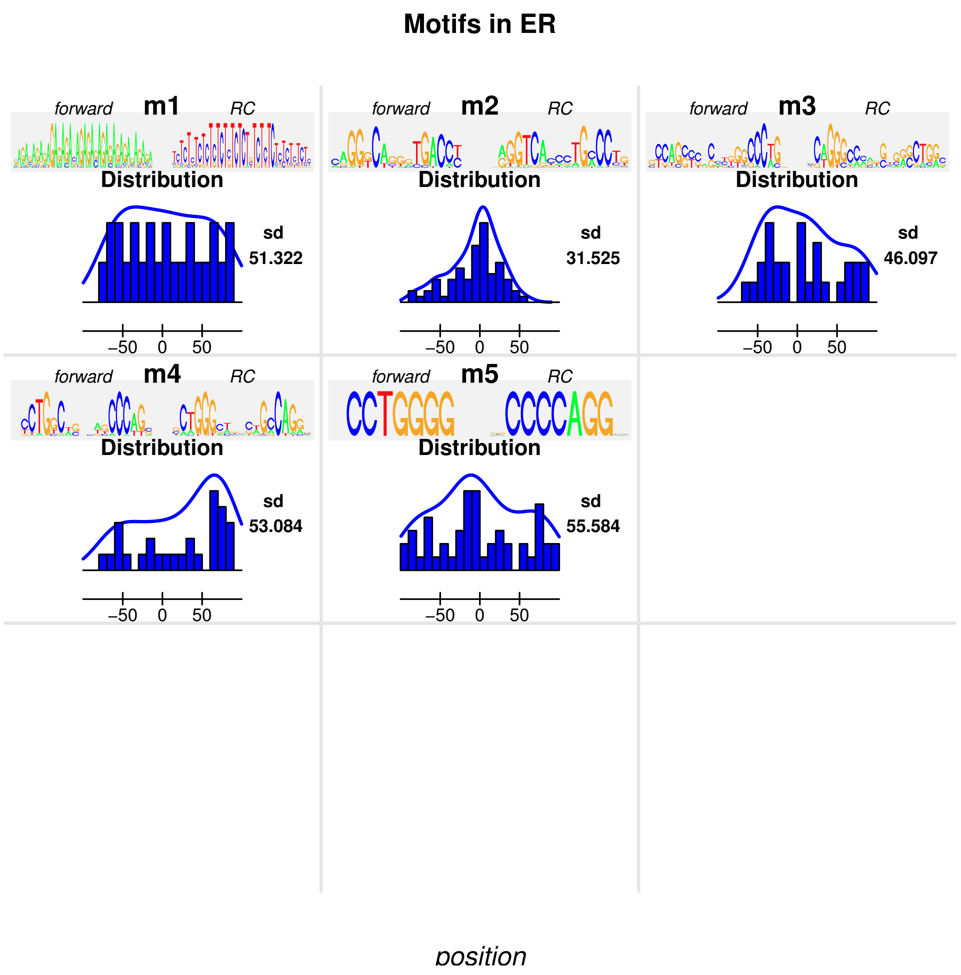
```
> library(rGADEM)
> library(BSgenome.Hsapiens.UCSC.hg18)
> RDfixed<-makeRangedDataOutput(pics, type="fixed", filter=myFilter)
> ERgadem<-GADEM(RDfixed,seed=1,genome=Hsapiens,verbose=TRUE)
> # save(ERgadem,file="ERgadem.rda")
```

Now we will post-process and visualize our regulatory motifs using the package `MotIV`. `MotIV` is similar to STAMP[?] with added graphics functionalities. It allows you to compare your identified motifs to known motifs according to some database (e.g. Jaspar[?]) and report the best matches. `MotIV` includes the Jaspar database and will use it by default, however you are free to use your own, please refer to the vignette of the `MotIV` for more details.

```
> library(MotIV)
> data(ERgadem)
> # Find the 5 best match in Jaspar.
> jaspar.match <- motifMatch(inputPWM = getPWM(ERgadem), top = 5)

        Ungapped Alignment
        Scores read
        Database read
        Motif matches : 5

> # Plot the motifs with their matches
> plot(jaspar.match , main = "Motifs in ER",top=5)
```



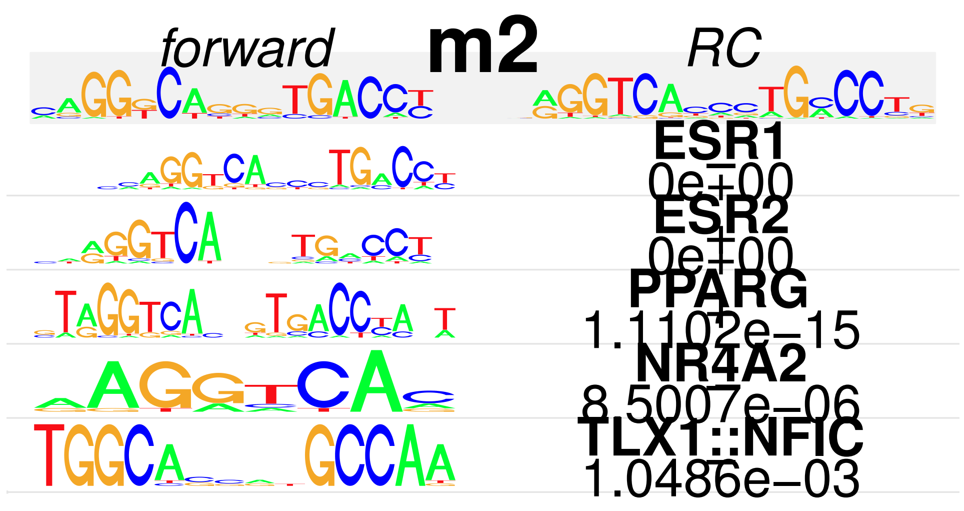
```
> library(MotIV)
> data(ERgadem)
> # Find the 5 best match in Jaspar.
> jaspar.match <- motifMatch(inputPWM = getPWM(ERgadem), top = 5)

        Ungapped Alignment
        Scores read
        Database read
        Motif matches : 5

> # Plot the motifs with their matches and distribution
> plot(jaspar.match , ERgadem,main = "Motifs in ER")
```



When rGADEM identifies multiple motifs, visualization and validation can become difficult. `MotIV` allows the user to filter motifs based on a set of filters. This is done with the function *setFilter*, as follows,

```
> Filter<-setFilter(name = "", tfname = "ESR1", top = 5, evalueMax = 10^-4)
> jaspar.match.ESR1<-filter(jaspar.match, Filter, verbose = TRUE)
> plot(jaspar.match.ESR1 , main = "ER motif",top=5)
> ERoc<-exportAsRangedData(jaspar.match.ESR1, ERgadem, correction=TRUE)
```

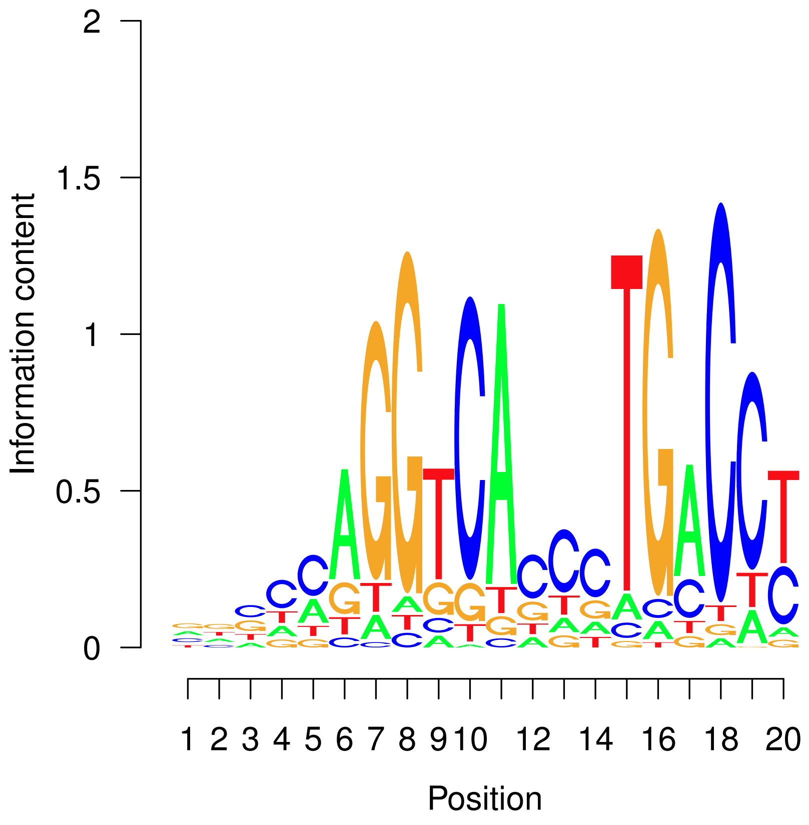
ER motif



rGADEM provides an option where the algorithm is initialized using a known motif, we refer to this option as a seeded analysis. Seeded analysis can be of particular use for short motifs and/or noisy data (e.g. Chip-chip) where a regular (unseeded) analysis might be more difficult. In the code snippet below, we use the ESR1 Jaspar motif as our seed.

The ER is included in the Jaspar database (named ESR1).

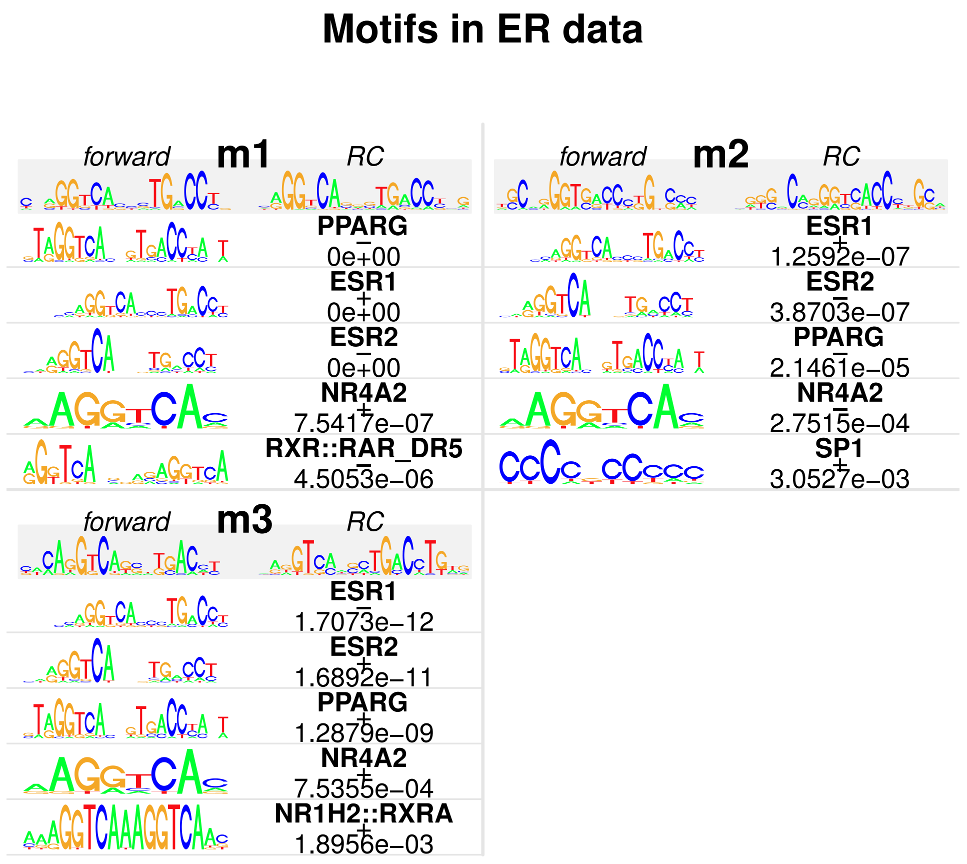
```
> # Normalize the PSSM -> PWM
> ERpwm<-apply(jaspar[["ESR1"]],2,function(x){x/sum(x)})
> # Make a PWM object that can be plotted with seqLogo
> library(seqLogo)
> ERpwmLogo<-makePWM(ERpwm)
> # Plot provided by seqLogo
> plot(ERpwmLogo)
```



```
> # Seeded analysis
> library(rGADEM)
> library(BSgenome.Hsapiens.UCSC.hg18)
> RDfixed<-makeRangedDataOutput(pics, type="fixed", filter=myFilter)
> ERgadem<-GADEM(RDfixed,genome=Hsapiens,pValue=5*10^-6,minSites=5,
+ Spwm=list(ERpwm,ERpwm,ERpwm,ERpwm,ERpwm,ERpwm,ERpwm,ERpwm))
```

Note that seeded runs are typically faster than unseeded ones. Now, we once again analyze the results using MotIV,

```
> jaspar.match <- motifMatch(inputPWM = getPWM(ERgadem), top = 5)

        Ungapped Alignment
        Scores read
        Database read
        Motif matches : 5

> plot(jaspar.match , main = "Motifs in ER data", top=5)
```



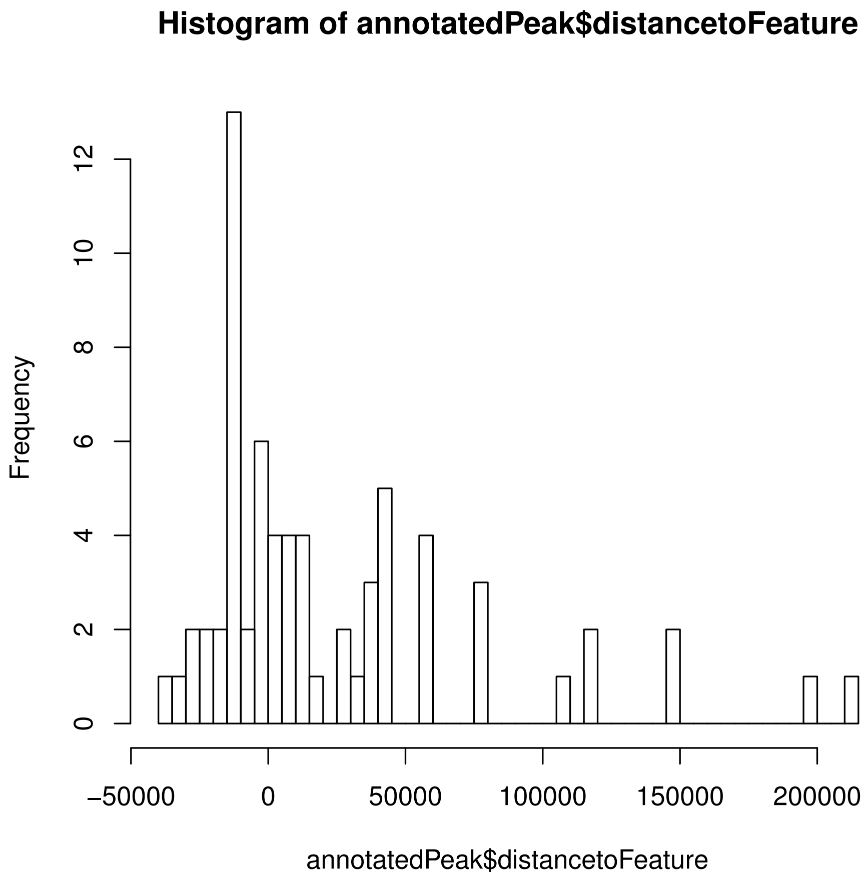
and we can see that one motif has been selected based on its match to the ESR1 Jaspar motif.

Exercise 5.1 *Try the above with a different eValue filter and/or motif name.*

6 Annotation of enriched regions:

Here we explore the package `ChIPpeakAnno`(?), a package that facilitates the batch annotation of the peaks identified from either ChIP-seq or ChIP-chip experiments. Using `ChIPpeakAnno` you can find the nearest gene, exon, miRNA or custom features supplied by users such as most conserved elements and other transcription factor binding sites leveraging the `IRanges` package. Here we load the TSS NCBI36 coordinates and look at possible possible overlaps with our ER binding sites. The *annotatePeakInBatch* will do that for you and in addition will compute the distance to the closest feature (here TSS).

```
> library(ChIPpeakAnno)
> data(TSS.human.NCBI36)
> annotatedPeak <- annotatePeakInBatch(ERoc, AnnotationData = TSS.human.NCBI36)
> # Plot the distances to TSS
> hist(annotatedPeak$distancetoFeature,50)
```



Exercise 6.1 *Use the* `rtracklayer` *package to plot the motif occurrences.*

Exercise 6.2 *Perform the same analysis with the FOXA1 data.*

```
Ungapped Alignment
Scores read
Database read
Motif matches : 5
```

Part II

Appendix: Installing PICS and MoTiV

To build the `PICS` package from source, make sure that the following is present in your system:

- GNU Scientific Library (GSL)
- Basic Linear Algebra Subprograms (BLAS)

GSL can be downloaded at `http://www.gnu.org/software/gsl/`. In addition, the package uses BLAS to perform basic vector and matrix operations. Please go to `http://www.netlib.org/blas/faq.html#5` for a list of optimized BLAS libraries for a variety of computer architectures. For instance, Mac users may use the built-in vecLib framework, while users of Intel machines may use the Math Kernel Library (MKL). A C compiler is needed to build the package as the core of the `PICS` function is coded in C.

For the package to be installed properly you might have to type the following command before installation:

```
export LD_LIBRARY_PATH='/path/to/GSL/:/path/to/BLAS/':$LD_LIBRARY_PATH
```

which will tell **R** where your GSL and BLAS libraries (see below for more details about BLAS libraries) are. Note that this might have already been configured on your system, so you might not have to do so. In case you need to do it, you might consider copying and pasting the line in your `.bashrc` so that you do not have to do it every time.

Now you are ready to install the package:

```
R CMD INSTALL PICS_x.y.z.tar.gz
```

The package will look for a BLAS library on your system, and by default it will choose gslcblas, which is not optimized for your system. To use an optimized BLAS library, you can use the `--with-blas` argument which will be passed to the `configure.ac` file. For example, on a Mac with vecLib pre-installed the package may be installed via:

```
R CMD INSTALL PICS_x.y.z.tar.gz --configure-args="--with-blas='-framework
vecLib'"
```

On a 64-bit Intel machine which has MKL as the optimized BLAS library, the command may look like:

```
R CMD INSTALL PICS_x.y.z.tar.gz --configure-args="- -with-blas='-L/usr/local/mkl/lib/em
-lmkl -lguide -lpthread'"
```

where `/usr/local/mkl/lib/em64t/` is the path to MKL.

If you prefer to install a prebuilt binary, you need GSL for successful installation.