The ChIPanalyser User's Guide

Patrick Martin

22/08/2017

Introduction

Transcriptional regulation is undeniably a key aspect of cellular homeostasis. It comes to no surprise that modern molecular biology and genomics have showed a keen interest in the subject. Transcription factors (TF) are a force to be reckoned with in the world of transcriptional regulation. Transcription factors are proteins that bind to DNA in a site-specific manner. Experimentally, this binding site can be determined by various methods such as SELEX-seq, EMSA or DNAse footprinting. The final result will be a sequence to which a given TF will bind preferentially. In many case, these results are presented in the form of a Position Frequency Matrix or Position Weight Matrix. However at a genome wide scale, modern molecular biology relies on methods such as Chromatin Immuno-precipitation linked to sequencing. This method generates a genome wide profile with peaks at sites of high TF occupancy. These experiments may be very costly and it would be interesting to be able to predict TF occupancy sites *in silico*. With this idea in mind, we present **ChIPanalyser** , a R package developed in the effort of understanding Transcription factor binding. At the core of this package resides an approximation of statistical thermodynamcis as suggested by Zabet (Zabet et al. 2015). The statistical thermodynamcis framework proposed by Zabet offers a strong ground for binding site prediction as it requires minimal data input. In its current version, ChIPAnalyser requires a DNA sequence, a Position Weight Matrix, the number of bound molecules (or TFs bound to DNA) and a scaling factor for TF specificity. To improve the accuracy of the model, it is also possible to incorporate DNA accessibility data.

Methods

As described above, ChIPAnalyser is based on an approximation of statistical thermodynamics. The core formula describing TF binding is given by :

$$P(N,a,\lambda,\omega)_j = \frac{N \cdot a_j \cdot e^{(\frac{1}{\lambda}\cdot\omega_j)}}{N \cdot a_j \cdot e^{(\frac{1}{\lambda}\cdot\omega_j)} + L \cdot n \cdot [a_i \cdot e^{(\frac{1}{\lambda}\cdot\omega_j)}]_i}$$

with

- N , the number of TF molecules bound to DNA
- a , DNA accessibility
- λ , a parameter scaling the specificity of a given TF
- ω , a Position Weight Matrix.

Work Flow - Quick start

Example data Loading

Before going through the inner workings of the package and the work flow, this section will quickly demonstrate how to load example datasets stored in the package. This data represents a minimal workable examples for the different functions. All data is derived from real biological data in *Drosophila melanogaster* (The *Drosophila melanogaster* genome can be found as a `BSgenome`).

```
library(ChIPanalyser)

#Load data
data(ChIPanalyserData)


# Loading DNASequenceSet from BSgenome object

library(BSgenome.Dmelanogaster.UCSC.dm3)

DNASequenceSet <-getSeq(BSgenome.Dmelanogaster.UCSC.dm3)



#Loading Position Frequency Matrix

PFM <- file.path(system.file("extdata",package="ChIPanalyser"),"BCDSlx.pfm")

#Checking if correctly loaded
ls()
```

```
## [1] "Access"          "DNASequenceSet" "PFM"            "eveLocus"
## [5] "eveLocusChip"    "geneRef"
```

The global environment should now contain a few new variables: DNASequenceSet,PFM,Access,geneRef, eveLocus, eveLocusChip.

- `DNASequenceSet` is DNAStringSet extracted from the *Drosophila melanogaster* genome (BSgenome). It is advised to use a full genome sequence for this object.
- `PFM` is a path to file. In this case, it is a Position Frequency Matrix derived from the Bicoid Transcription factor in *Drosophila melanogaster*. This PFM is in *raw* format. Although it is possible to to directly use a PFM R matrix (see motifDB R package), we chose to use a path to a file for this example. Most PFM's found online will come in a text file (with various formats: RAW, TRANSFAC, JASPAR). `ChIPanalyser` is capable of handling all these formats and parsing these files to usable objects within the package.
- `Access` is a `GRanges` object containing accessible DNA for the sequence above.
- `geneRef` is a `GRanges` containing genetic information (exon, intron, 3'UTR, 5'UTR) for the sequence above.
- `eveLocus` is a `GRanges` object with genomic postion for the eve strip locus in *Drosophila melanogaster*.
- `eveLocusChip` is a data frame with ChIP score in the format of a simple bed file (4 columns : chromosome, start, end and score) for Bicoid transcription factor.

Quick Start

This section presents a quick work flow. For details on the work flow and objects, see section **Work Flow - Full Guide**

Step 1 - Building Data objects and Pre-processing ChIP data

The first step is to set up your data storing objects and extract normalised ChIP scores at loci of interest. When provided with a PFM, the `genomicProfileParameters` object will automatically convert it to a

PWM. The `occupancyProfileParameters` object requires parameters extracted from ChIP data. If using the `processingChIPseq` function, the `occupancyProfileParameters` will be generated internally.

```
# Building a genomicProfileParameters objects for data
# storage and PWM computation
GPP <- genomicProfileParameters(PFM=PFM,PFMFormat="raw",
    BPFrequency=DNASequenceSet,
    ScalingFactorPWM = 1.5,
    PWMThreshold = 0.7)
GPP
```

```
## Object Class:genomicProfileParameters
##

##
## PWM:

##          [,1]       [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## A -0.09520642 -1.0929970 -4.170092  1.761696  1.761696 -5.263560 -9.445015
## C  0.55082162  0.8819112 -4.550984 -9.445015 -9.445015 -9.445015  2.258075
## G  0.63156095 -2.0457025 -9.445015 -4.333846 -4.333846 -3.164873 -9.445015
## T -1.57041086  0.5565425  1.743852 -9.445015 -9.445015  1.735331 -9.445015
##        [,8]
## A -4.451342
## C  2.091309
## G -3.573736
## T -1.875062

##
## PFM:

##   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## A  190   95   11  689  689    5    0    9
## C  213  268    6    0    0    0  696  620
## G  225   35    0    7    7   16    0   12
## T   68  298  679    0    0  675    0   55

##
## PFMFormat: raw

##
## PWM Scores at Sites higher than Threshold:

## Warning in showList(object, showFunction, print.classinfo = TRUE): Note that starting with BioC 3.7,
##   GRangesList **instances** needs to be set to
##   "CompressedGRangesList". Please update this object with
##   'updateObject(object, verbose=TRUE)' and re-serialize it.

## GRangesList object of length 0:
## <0 elements>
##
## -------
## seqinfo: no sequences

##
## No Accessible DNA at Loci:

##

##
```

```
## Genomic Profile Parameters:

## Lambda:  1.5
## BP Frequency:    0.2916399   0.2088135   0.2085611   0.2909855

## Pseudocount: 1
## Natural log: FALSE
## Number Of Sites: 0
## maxPWMScore:
## minPWMScore:
## PWMThreshold: 0.7

## Average Exponential PWM Score:

## DNA Sequence Length:
## Strand Rule: max
## Strand: +-
```

```
# Building occupancyProfileParameters with default values
OPP <- occupancyProfileParameters()
OPP
```

```
## Object Class:occupancyProfileParameters
##

## Ploidy: 2

## boundMolecules:  1000

## backgroundSignal: 0
## maxSignal: 1
## chipMean: 150
## chipSd: 150
## chipSmooth: 250
## Step Size: 10
```

```
# Building occupancyProfileParameters with custom values
OPP <- occupancyProfileParameters(ploidy= 2,
    boundMolecules= 1000,
    chipMean = 200,
    chipSd = 200,
    chipSmooth = 250,
    maxSignal = 1.847,
    backgroundSignal = 0.02550997)
OPP
```

```
## Object Class:occupancyProfileParameters
##
## Ploidy: 2

## boundMolecules:  1000

## backgroundSignal: 0.02550997
## maxSignal: 1.847
## chipMean: 200
## chipSd: 200
## chipSmooth: 250
## Step Size: 10
```

```
## Extracting ChIP score
eveLocusChip<-processingChIPseq(eveLocusChip,eveLocus,noiseFilter="zero",cores=1)
str(eveLocusChip)
```

```
## List of 2
##  $ :List of 1
##   ..$ eve: num [1:16000] 0.0108 0.0108 0.0108 0.0108 0.0108 ...
##  $ :Formal class 'occupancyProfileParameters' [package "ChIPanalyser"] with 9 slots
##   .. ..@ ploidy          : num 2
##   .. ..@ boundMolecules  : num 1000
##   .. ..@ backgroundSignal: num 0.0915
##   .. ..@ maxSignal       : num 1
##   .. ..@ chipMean        : num 150
##   .. ..@ chipSd          : num 150
##   .. ..@ chipSmooth      : num 250
##   .. ..@ stepSize        : num 10
##   .. ..@ removeBackground: num 0
```

```
### Extracting occupancy profile parameters object built from ChIP data
OPP<-eveLocusChip[[2]]
eveLocusChip<-eveLocusChip[[1]]
```

Step 2 - Optimal Parameters

The model is based on the approximation of statistical thermodynamics with inference of two parameters (ScalingFactorPWM and boundMolecules). In order to infer these parameters, we suggest to use `computeOptimal`. Values that should be tested for `ScalingFactorPWM` and for `boundMolecules` should be provided by user as described above. If these values are not provided (default value OR only one value for each parameter), then they will be assigned internally. ChIPanalyser also has multi-core support. If you are using large genomes, using multiple cores will significantly decrease computational time. The internal values are the following:

```
ScalingFactorPWM(genomicProfileParameters) <- c(0.25, 0.5, 0.75, 1, 1.25,
    1.5, 1.75, 2, 2.5, 3, 3.5 ,4 ,4.5, 5)

boundMolecules(occupancyProfileParameters) <- c(1, 10, 20, 50, 100,
    200, 500,1000,2000, 5000,10000,20000,50000, 100000,
    200000, 500000, 1000000)
```

`computeOptimal`contains the following arguments:

```
optimalParam <- suppressWarnings(computeOptimal(DNASequenceSet = DNASequenceSet,
    genomicProfileParameters = GPP,
    LocusProfile = eveLocusChip,
    setSequence = eveLocus,
    DNAAccessibility = Access,
    occupancyProfileParameters = OPP,
    optimalMethod = "all",
    peakMethod="moving_kernel",
    cores=1))
```

```
## Computing Genome Wide PWM Score

## Computing PWM Score at Loci & Extracting Sites Above Threshold

## Single Core PWM Scores Extraction

## Computing Occupancy
```

```
## Computing ChIP-seq-like Profile
```

```
## Computing Accuracy of Profile
```

```
str(optimalParam)
```

```
## List of 3
##  $ Optimal Parameters:List of 12
##   ..$ pearsonMean  : Named chr [1:2] "1" "20000"
##   .. ..- attr(*, "names")= chr [1:2] "ScalingFactor" "BoundMolecules"
##   ..$ spearmanMean : Named chr [1:2] "1.25" "1000"
##   .. ..- attr(*, "names")= chr [1:2] "ScalingFactor" "BoundMolecules"
##   ..$ kendallMean  : Named chr [1:2] "1.25" "1000"
##   .. ..- attr(*, "names")= chr [1:2] "ScalingFactor" "BoundMolecules"
##   ..$ MSEMean      : Named chr [1:2] "1" "50000"
##   .. ..- attr(*, "names")= chr [1:2] "ScalingFactor" "BoundMolecules"
##   ..$ ksMean       : Named chr [1:2] "1.75" "200"
##   .. ..- attr(*, "names")= chr [1:2] "ScalingFactor" "BoundMolecules"
##   ..$ geometricMean: Named chr [1:2] "1" "20000"
##   .. ..- attr(*, "names")= chr [1:2] "ScalingFactor" "BoundMolecules"
##   ..$ precisionMean: Named chr [1:2] "0.5" "20000"
##   .. ..- attr(*, "names")= chr [1:2] "ScalingFactor" "BoundMolecules"
##   ..$ recallMean   : Named chr [1:2] "0.5" "2e+05"
##   .. ..- attr(*, "names")= chr [1:2] "ScalingFactor" "BoundMolecules"
##   ..$ FscoreMean   : Named chr [1:2] "0.5" "20000"
##   .. ..- attr(*, "names")= chr [1:2] "ScalingFactor" "BoundMolecules"
##   ..$ AccuracyMean : Named chr [1:2] "0.5" "20000"
##   .. ..- attr(*, "names")= chr [1:2] "ScalingFactor" "BoundMolecules"
##   ..$ MCCMean      : Named chr [1:2] "0.5" "20000"
##   .. ..- attr(*, "names")= chr [1:2] "ScalingFactor" "BoundMolecules"
##   ..$ AUCMean      : Named chr [1:2] "0.5" "20000"
##   .. ..- attr(*, "names")= chr [1:2] "ScalingFactor" "BoundMolecules"
##  $ Optimal Matrix    :List of 12
##   ..$ pearsonMean  : num [1:14, 1:17] 0.802 0.806 0.805 0.689 0.71 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:14] "0.25" "0.5" "0.75" "1" ...
##   .. .. ..$ : chr [1:17] "1" "10" "20" "50" ...
##   ..$ spearmanMean : num [1:14, 1:17] 0.632 0.626 0.632 0.658 0.585 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:14] "0.25" "0.5" "0.75" "1" ...
##   .. .. ..$ : chr [1:17] "1" "10" "20" "50" ...
##   ..$ kendallMean  : num [1:14, 1:17] 0.468 0.462 0.465 0.496 0.442 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:14] "0.25" "0.5" "0.75" "1" ...
##   .. .. ..$ : chr [1:17] "1" "10" "20" "50" ...
##   ..$ MSEMean      : num [1:14, 1:17] 1.395 1.522 1.315 0.78 0.855 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:14] "0.25" "0.5" "0.75" "1" ...
##   .. .. ..$ : chr [1:17] "1" "10" "20" "50" ...
##   ..$ ksMean       : num [1:14, 1:17] 0.699 0.713 0.666 0.417 0.518 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:14] "0.25" "0.5" "0.75" "1" ...
##   .. .. ..$ : chr [1:17] "1" "10" "20" "50" ...
##   ..$ geometricMean: num [1:14, 1:17] 12.17 13.82 11.23 4.71 5.76 ...
##   .. ..- attr(*, "dimnames")=List of 2
```

```
##   .. .. ..$ : chr [1:14] "0.25" "0.5" "0.75" "1" ...
##   .. .. ..$ : chr [1:17] "1" "10" "20" "50" ...
##   ..$ precisionMean: num [1:14, 1:17] 0.384 0.382 0.369 0.346 0.342 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:14] "0.25" "0.5" "0.75" "1" ...
##   .. .. ..$ : chr [1:17] "1" "10" "20" "50" ...
##   ..$ recallMean   : num [1:14, 1:17] 0.787 0.785 0.793 0.794 0.781 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:14] "0.25" "0.5" "0.75" "1" ...
##   .. .. ..$ : chr [1:17] "1" "10" "20" "50" ...
##   ..$ FscoreMean   : num [1:14, 1:17] 0.347 0.345 0.34 0.328 0.323 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:14] "0.25" "0.5" "0.75" "1" ...
##   .. .. ..$ : chr [1:17] "1" "10" "20" "50" ...
##   ..$ AccuracyMean : num [1:14, 1:17] 0.636 0.636 0.611 0.562 0.557 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:14] "0.25" "0.5" "0.75" "1" ...
##   .. .. ..$ : chr [1:17] "1" "10" "20" "50" ...
##   ..$ MCCMean      : num [1:14, 1:17] 0.29 0.288 0.277 0.241 0.224 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:14] "0.25" "0.5" "0.75" "1" ...
##   .. .. ..$ : chr [1:17] "1" "10" "20" "50" ...
##   ..$ AUCMean      : num [1:14, 1:17] 0.89 0.889 0.886 0.862 0.84 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:14] "0.25" "0.5" "0.75" "1" ...
##   .. .. ..$ : chr [1:17] "1" "10" "20" "50" ...
##  $ method            : chr "all"
```

This Function might take some time to compute. Do not be alarmed if it takes some time to run. You should be notified of the progress of the function as it goes

This function is a combination of all the functions bellow with some more magic to it. In the following steps we will describe each of the functions.

Step 3 - Genome Wide Scoring

Computing Genome Wide metrics that will be used further down the line. It is possible to set a higher number of cores to decrease computational time.

```
genomeWide <- computeGenomeWidePWMScore(DNASequenceSet=DNASequenceSet,
    genomicProfileParameters=GPP, DNAAccessibility = Access,cores=1)
```

```
## Scoring whole genome
```

```
## Accessible DNA ~ Both strands
```

```
## Computing Mean waiting time
```

```
genomeWide
```

```
## Object Class:genomicProfileParameters
##
```

```
##
## PWM:
```

```
##          [,1]       [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## A -0.09520642 -1.0929970 -4.170092  1.761696  1.761696 -5.263560 -9.445015
```

```
## C  0.55082162  0.8819112 -4.550984 -9.445015 -9.445015 -9.445015  2.258075
## G  0.63156095 -2.0457025 -9.445015 -4.333846 -4.333846 -3.164873 -9.445015
## T -1.57041086  0.5565425  1.743852 -9.445015 -9.445015  1.735331 -9.445015
##        [,8]
## A -4.451342
## C  2.091309
## G -3.573736
## T -1.875062

##
## PFM:

##   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## A  190   95   11  689  689    5    0    9
## C  213  268    6    0    0    0  696  620
## G  225   35    0    7    7   16    0   12
## T   68  298  679    0    0  675    0   55

##
## PFMFormat: raw

##
## PWM Scores at Sites higher than Threshold:

## Warning in showList(object, showFunction, print.classinfo = TRUE): Note that starting with BioC 3.7,
##   GRangesList **instances** needs to be set to
##   "CompressedGRangesList". Please update this object with
##   'updateObject(object, verbose=TRUE)' and re-serialize it.

## GRangesList object of length 0:
## <0 elements>
##
## -------
## seqinfo: no sequences

##
## No Accessible DNA at Loci:

##

##
## Genomic Profile Parameters:

## Lambda:  1.5
## BP Frequency:    0.2916399   0.2088135   0.2085611   0.2909855

## Pseudocount: 1
## Natural log: FALSE
## Number Of Sites: 0
## maxPWMScore: 12.8654303345745
## minPWMScore: -49.2286544334621
## PWMThreshold: 0.7

## Average Exponential PWM Score:   0.8457538

## DNA Sequence Length: 3145351
## Strand Rule: max
## Strand: +-
```

`computeGenomeWidePWMScore` will return a `genomicProfileParameters` object with updated values for `maxPWMScore`, `minPWMScore`,`averageExpPWMScore`, and `DNASequenceLength`.

Step 4 - PWM Scores Above Threshold

Once genome wide scores have been computed, the `genomeWide` object (previously computed) should be parsed to the next function. The next function will compute sites above the assigned threshold (see below) for a given locus (or set of loci). If no Locus is provided then the whole genome will be considered. It is possible to set a higher number of cores to decrease computational time.

** It is important to set names to your setSequence object (see below). We recommend to set names yourself to make your analysis easier to keep track of. Names will be set internally for computational reasons but there must be concordance between setSequence and ChIP data **

** This aspect cannot be stressed enough**

```
SitesAboveThreshold <- computePWMScore(DNASequenceSet=DNASequenceSet,
    genomicProfileParameters=genomeWide,
    setSequence=eveLocus, DNAAccessibility = Access,cores=1)
```

```
## Single Core PWM Scores Extraction
```

```
SitesAboveThreshold
```

```
## Object Class:genomicProfileParameters
##
```

```
##
## PWM:
```

```
##          [,1]       [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## A -0.09520642 -1.0929970 -4.170092  1.761696  1.761696 -5.263560 -9.445015
## C  0.55082162  0.8819112 -4.550984 -9.445015 -9.445015 -9.445015  2.258075
## G  0.63156095 -2.0457025 -9.445015 -4.333846 -4.333846 -3.164873 -9.445015
## T -1.57041086  0.5565425  1.743852 -9.445015 -9.445015  1.735331 -9.445015
##        [,8]
## A -4.451342
## C  2.091309
## G -3.573736
## T -1.875062
```

```
##
## PFM:
```

```
##   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## A  190   95   11  689  689    5    0    9
## C  213  268    6    0    0    0  696  620
## G  225   35    0    7    7   16    0   12
## T   68  298  679    0    0  675    0   55
```

```
##
## PFMFormat: raw
```

```
##
## PWM Scores at Sites higher than Threshold:
```

```
## GRangesList object of length 1:
## $eve
## GRanges object with 420 ranges and 2 metadata columns:
##       seqnames          ranges strand |          PWMScore DNAaffinity
##          <Rle>       <IRanges>  <Rle> |         <numeric>   <numeric>
##   eve    chr2R 5860705-5860712      + | -1.84573024098586           1
##   eve    chr2R 5860709-5860716      + | -4.96148500199546           1
```

```
##   eve    chr2R 5860715-5860722      + |  8.81832070896316           1
##   eve    chr2R 5860728-5860735      + |  4.24981127739825           1
##   eve    chr2R 5860758-5860765      + | -5.25856937621247           1
##   ...      ...             ...    ... .                ...         ...
##   eve    chr2R 5876629-5876636      + |  5.76325435176529           1
##   eve    chr2R 5876635-5876642      + | 0.824810948340001           1
##   eve    chr2R 5876641-5876648      - |  -5.0584607351313           1
##   eve    chr2R 5876666-5876673      + |  1.87745682827728           1
##   eve    chr2R 5876684-5876691      + | -2.38839005613713           1
##
## -------
## seqinfo: 1 sequence from an unspecified genome; no seqlengths

##
## No Accessible DNA at Loci:

## -

##
## Genomic Profile Parameters:

## Lambda:  1.5
## BP Frequency:    0.2916399   0.2088135   0.2085611   0.2909855

## Pseudocount: 1
## Natural log: FALSE
## Number Of Sites: 0
## maxPWMScore: 12.8654303345745
## minPWMScore: -49.2286544334621
## PWMThreshold: 0.7

## Average Exponential PWM Score:   0.8457538

## DNA Sequence Length: 3145351
## Strand Rule: max
## Strand: +-
```

This function returns another `genomicProfileParameters` object with an updated `AllSitesAboveThreshold` slot. This slot contains a `GRanges` object with sites above threshold and associated PWMScores.

Step 4 - compute Occupancy

From the PWMScores, `ChIPanalyser` will compute occupancy for each sites above threshold.

```
Occupancy <- computeOccupancy(SitesAboveThreshold,
    occupancyProfileParameters= OPP)
```

```
## Computing Occupancy at sites higher than threshold.
```

```
Occupancy
```

```
## Object Class:genomicProfileParameters
##

##
## PWM:

##          [,1]       [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## A -0.09520642 -1.0929970 -4.170092  1.761696  1.761696 -5.263560 -9.445015
## C  0.55082162  0.8819112 -4.550984 -9.445015 -9.445015 -9.445015  2.258075
```

```
## G  0.63156095 -2.0457025 -9.445015 -4.333846 -4.333846 -3.164873 -9.445015
## T -1.57041086  0.5565425  1.743852 -9.445015 -9.445015  1.735331 -9.445015
##        [,8]
## A -4.451342
## C  2.091309
## G -3.573736
## T -1.875062

##
## PFM:

##   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## A  190   95   11  689  689    5    0    9
## C  213  268    6    0    0    0  696  620
## G  225   35    0    7    7   16    0   12
## T   68  298  679    0    0  675    0   55

##
## PFMFormat: raw

##
## PWM Scores at Sites higher than Threshold:

## $`lambda = 1.5 & boundMolecules = 1000`
## GRangesList object of length 1:
## $eve
## GRanges object with 420 ranges and 3 metadata columns:
##       seqnames          ranges strand |          PWMScore DNAaffinity
##          <Rle>       <IRanges>  <Rle> |         <numeric>   <numeric>
##   eve    chr2R 5860705-5860712      + | -1.84573024098586           1
##   eve    chr2R 5860709-5860716      + | -4.96148500199546           1
##   eve    chr2R 5860715-5860722      + |  8.81832070896316           1
##   eve    chr2R 5860728-5860735      + |  4.24981127739825           1
##   eve    chr2R 5860758-5860765      + | -5.25856937621247           1
##   ...      ...             ...    ... .                ...         ...
##   eve    chr2R 5876629-5876636      + |  5.76325435176529           1
##   eve    chr2R 5876635-5876642      + | 0.824810948340001           1
##   eve    chr2R 5876641-5876648      - |  -5.0584607351313           1
##   eve    chr2R 5876666-5876673      + |  1.87745682827728           1
##   eve    chr2R 5876684-5876691      + | -2.38839005613713           1
##               Occupancy
##               <numeric>
##   eve 0.0915185584072193
##   eve 0.0914749225700571
##   eve  0.148659402751388
##   eve 0.0943624008602243
##   eve 0.0914737995560999
##   ...                ...
##   eve  0.099361641959952
##   eve 0.0917645162456286
##   eve 0.0914745312764083
##   eve 0.0920652829761032
##   eve 0.0915034155828825
##
## -------
## seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

```
##
## No Accessible DNA at Loci:
## -
##
## Genomic Profile Parameters:
## Lambda:  1.5
## BP Frequency:    0.2916399   0.2088135   0.2085611   0.2909855
## Pseudocount: 1
## Natural log: FALSE
## Number Of Sites: 0
## maxPWMScore: 12.8654303345745
## minPWMScore: -49.2286544334621
## PWMThreshold: 0.7
## Average Exponential PWM Score:   0.8457538
## DNA Sequence Length: 3145351
## Strand Rule: max
## Strand: +-
```

This function will return a `genomicProfileParameters` object with an updated `AllSitesAboveThreshold`. Now the Occupancy values for each sites are included.

Step 5 - compute ChIP -seq like profiles

The ultimate goal of `ChIPanalyser` is to produce ChIP-seq like profile predicting transcription factor binding. To do so, the following function will compute ChIP-seq like scores from occupancy values.

```
chipProfile <- computeChipProfile(setSequence = eveLocus,
    occupancy = Occupancy,occupancyProfileParameters = OPP,
    method="moving_kernel")
```

```
## Computing ChIP Profile
```

```
chipProfile
```

```
## $`lambda = 1.5 & boundMolecules = 1000`
## $`lambda = 1.5 & boundMolecules = 1000`$eve
## GRanges object with 1600 ranges and 1 metadata column:
##       seqnames          ranges strand |               ChIP
##          <Rle>       <IRanges>  <Rle> |          <numeric>
##   eve    chr2R 5860693-5860703      * | 0.0633101738995645
##   eve    chr2R 5860703-5860713      * | 0.0686501364849457
##   eve    chr2R 5860713-5860723      * | 0.0741342776797116
##   eve    chr2R 5860723-5860733      * | 0.0797869804734581
##   eve    chr2R 5860733-5860743      * | 0.0856333772959523
##   ...      ...             ...    ... .                ...
##   eve    chr2R 5876643-5876653      * | 0.0800444269473348
##   eve    chr2R 5876653-5876663      * | 0.0762385945900301
##   eve    chr2R 5876663-5876673      * | 0.0723418265102848
##   eve    chr2R 5876673-5876683      * | 0.0683367973234721
##   eve    chr2R 5876683-5876693      * | 0.0642057003062555
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

This function will return a`List` of `GRangesLists` of `GRanges`. Each element of the list represents a combination of `ScalingFactorPWM` and `boundMolecules`. The `GRangesList` contains the Loci of interest. Finally, the individual `GRanges` contains ChIP-seq like scores for every n base pairs (with n = stepSize, see bellow).

This object may be difficult to navigate if many different parameters, or Loci are used. In order to facilitate navigation, we included a search function. **See function: searchSites** This function can also be used to navigate `AllSitesAboveThreshold` slot after occupancy scores have been computed.

Step 6 - Model Accuracy

Assessing model quality (predicted model against real ChIP-seq data).

```
AccuracyEstimate <- profileAccuracyEstimate(LocusProfile = eveLocusChip,
    predictedProfile = chipProfile, occupancyProfileParameters = OPP,method="all")
```

```
## Warning in ks.test(predicted, locusProfile): p-value will be approximate in
## the presence of ties
```

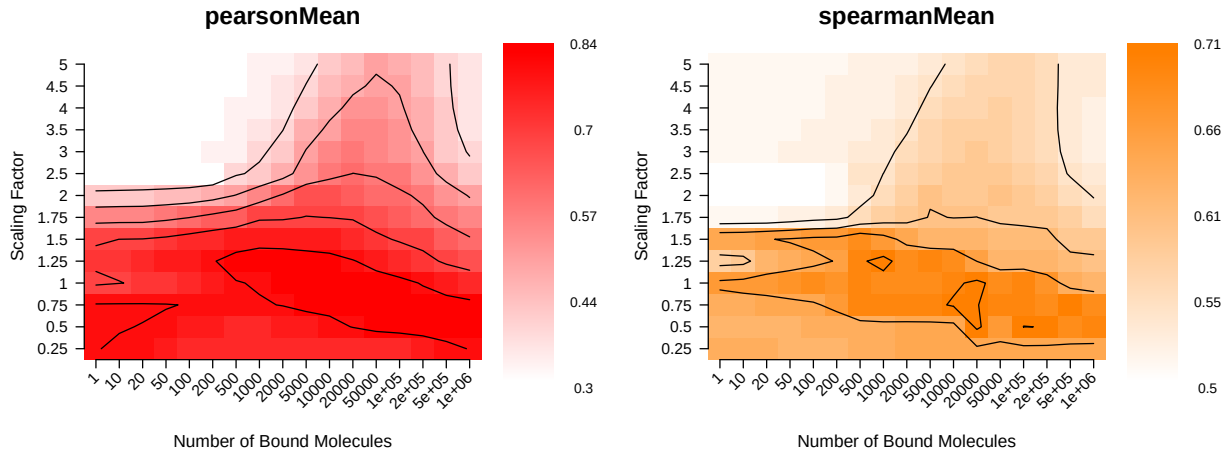
```
AccuracyEstimate <-AccuracyEstimate[[1]][[1]][[1]]
AccuracyEstimate
```

```
##           MSE       pearson      spearman       kendall        KsDist
##    0.01091931    0.77735490    0.66616368    0.50456610    0.19500000
##        KsPval     geometric   pearsonMean  spearmanMean   kendallMean
##    0.00000000    0.90072786    0.77735490    0.66616368    0.50456610
##       MSEMean        ksMean geometricMean precisionMean    recallMean
##    0.47768643    0.19500000    0.90072786    0.35149108    0.79913747
##    FscoreMean  AccuracyMean       MCCMean       AUCMean     precision
##    0.33277032    0.56294790    0.24380162    0.86504334    0.35149108
##        recall            f1      accuracy           MCC           AUC
##    0.79913747    0.33277032    0.56294790    0.24380162    0.86504334
```

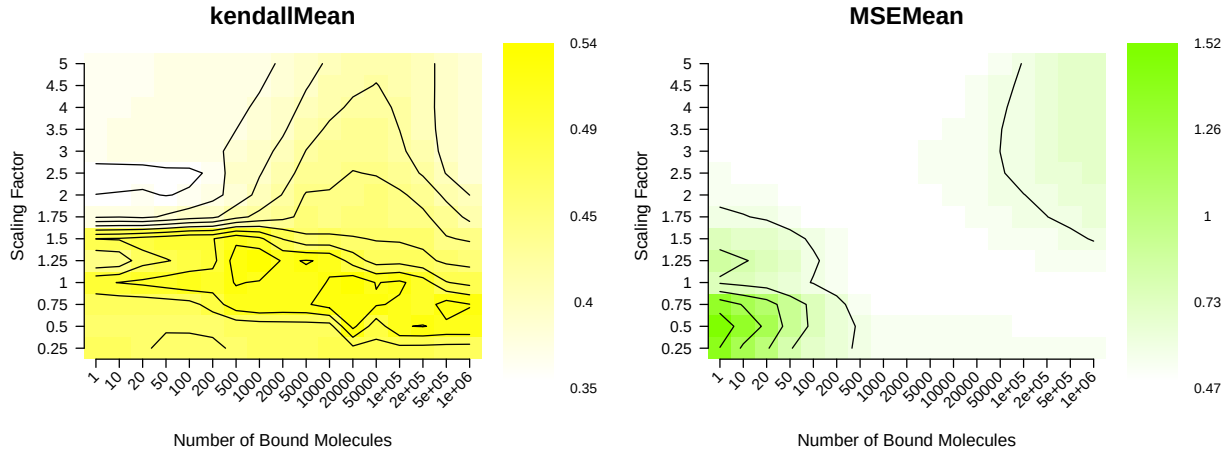
Step 7 - Plotting

Finally, once all has been computed, it is possible to plot the results.

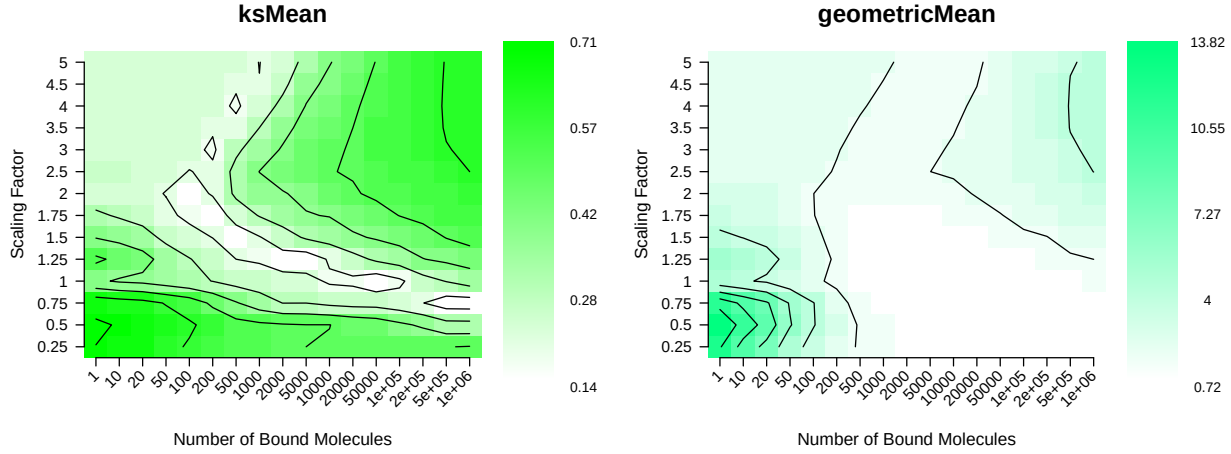
```
# Plotting Optimal heat maps
par(oma=c(0,0,3,0))
layout(matrix(1:8,ncol=4, byrow=T),width=c(6,1.5,6,1.5),height=c(1,1))
plotOptimalHeatMaps(optimalParam,layout=FALSE)
```


pearsonMean
Scaling Factor
Number of Bound Molecules
0.84
0.7
0.57
0.44
0.3

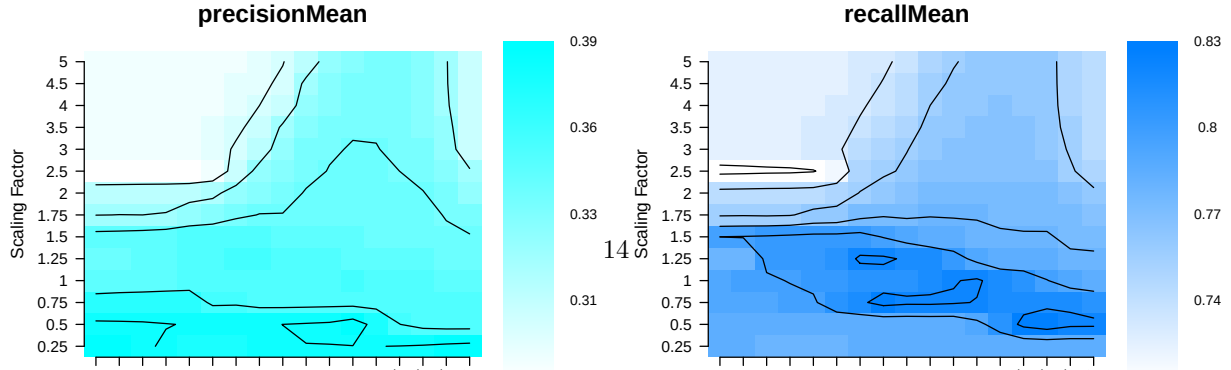
spearmanMean
Scaling Factor
Number of Bound Molecules
0.71
0.66
0.61
0.55
0.5


kendallMean
Scaling Factor
Number of Bound Molecules
0.54
0.49
0.45
0.4
0.35

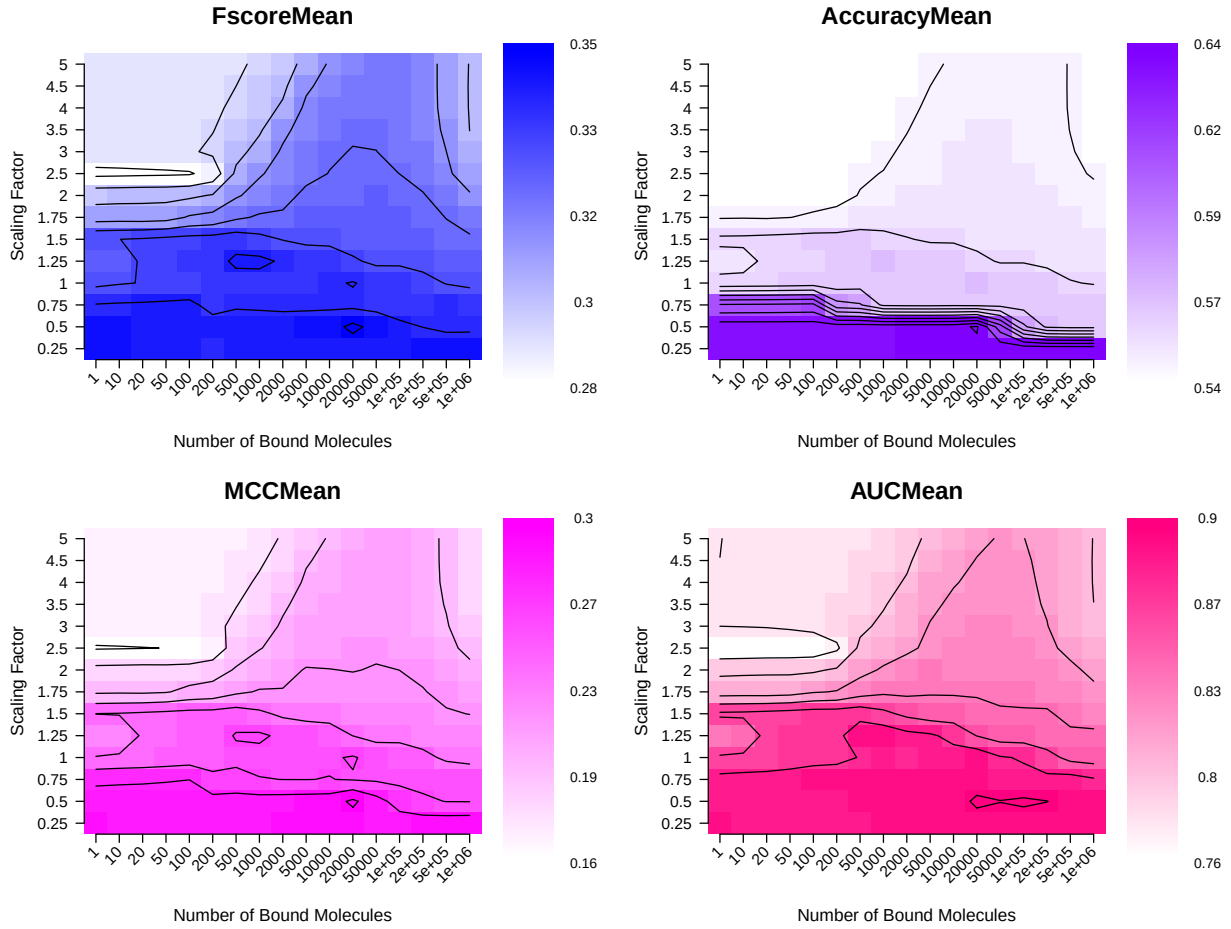
MSEMean
Scaling Factor
Number of Bound Molecules
1.52
1.26
1
0.73
0.47


ksMean
Scaling Factor
Number of Bound Molecules
0.71
0.57
0.42
0.28
0.14

geometricMean
Scaling Factor
Number of Bound Molecules
13.82
10.55
7.27
4
0.72

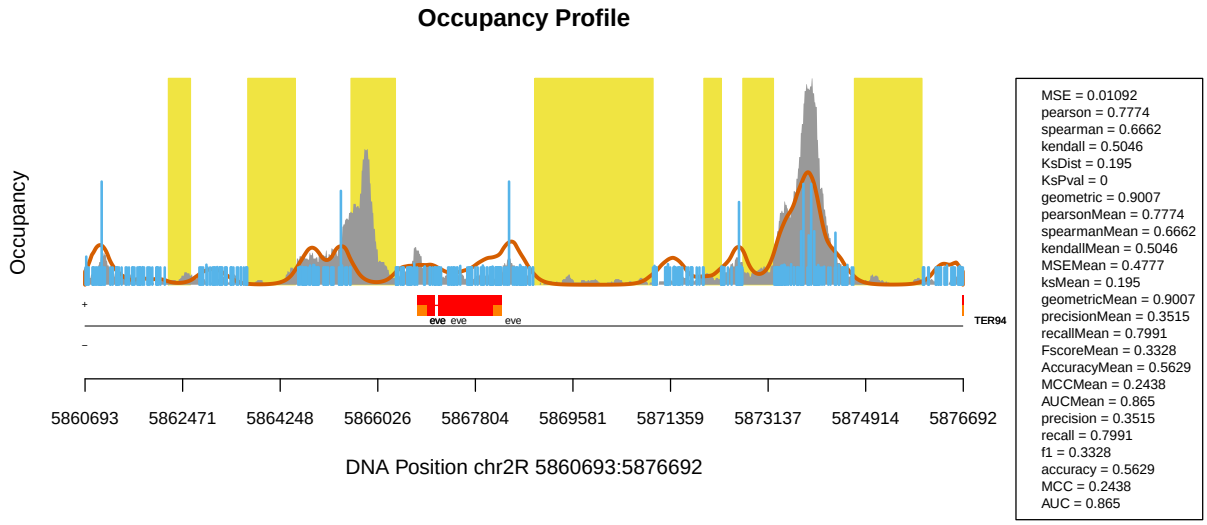

precisionMean
Scaling Factor
0.39
0.36
0.33
0.31

recallMean
Scaling Factor
0.83
0.8
0.77
0.74




```
# Plotting occupancy Profile
##

plotOccupancyProfile(predictedProfile=chipProfile[[1]][[1]],
    setSequence=eveLocus,
    chipProfile = eveLocusChip[[1]],
    DNAAccessibility = Access,
    occupancy = AllSitesAboveThreshold(Occupancy)[[1]][[1]],
    profileAccuracy=AccuracyEstimate,
    occupancyProfileParameters = OPP,
    geneRef=geneRef
    )
```



Work Flow - Full Guide

This section will describe `ChIPanalyser`'s work flow. However in this section we will describe in detail data objects, parameters, and functions. Please refer to this section if in doubt. If the doubt persists, don't hesitate to send an email to the maintainer.

Data objects - Genomic Profile Parameters

The very first aspect to consider when using ChIPAnalyser is data input. Many (if not all functions) require specific data inputs and parameters in order to carry out the computation. To facilitate, the storage of these parameters, we created a `genomicProfileParameters` object (S4 class). This is the very first step before any other work. All other functions rely on this `genomicProfileParameters` object in one form or another. The output of most functions will be a `genomicProfileParameters` object. Thus the output of one functions should be used as an input for the next functions in the pipeline. All functions are described bellow in section **Work Flow - Analysis**.

This object comes in the following form:

```
genomicProfileParameters(PWM, PFM, ScalingFactorPWM, PFMFormat, pseudocount,
    BPFrequency, naturalLog, noOfSites,
    minPWMScore, maxPWMScore, PWMThreshold,
    AllSitesAboveThreshold, DNASequenceLength,
    averageExpPWMScore, strandRule,whichstrand, NoAccess)
```

To build a `genomicProfileParameters` object :

```
# Assign Value wanted for each parameter
GPP <- genomicProfileParameters(PWM, PFM,ScalingFactorPWM, PFMFormat,
    pseudocount, BPFrequency, naturalLog, noOfSites,
    PWMThreshold, DNASequenceLength,
    strandRule, whichstrand)
```

As one can see, `genomicProfileParameters` contains many arguments. However many of these arguments already have default values assigned to them. Some of the arguments should not be set by user. These values are

computed internally and will automatically updated (minPWMScore, maxPWMScore, AllSitesAboveThreshold, NoAccess). In this situation, most arguments are not required to build a `genomicProfileParameters` object and a minimal build can be described as:

```
# return empty genomicProfileParameters object
GPP <- genomicProfileParameters()
# return minimal working object
GPP <- genomicProfileParameters(PFM=PFM,PFMFormat="raw")
# Suggested Minimal Build
GPP <- genomicProfileParameters(PFM=PFM,PFMFormat="raw",
BPFrequency=DNASequenceSet)
```

Although many parameters have assigned default values, it is recommended to use custom parameters to better fit the needs of the analysis. The method described above will build a new `genomicProfileParameters` object with the values that were assigned to each argument. Only three slots are required in order to build a `genomicProfileParameters` object (see below - **The compulsory ones**).Most other slots are optional. If after building `genomicProfileParameters`, you wish to modify the value of only *one* slot and keep the values that you had previously assigned, it is possible to modify each slot individually by using the slot *access/setter* methods. Each slot and it's *access/setter* method is described below.

Position Matricies - The compulsory ones

- `PWM` , a Position Weight Matrix. If a Position Weight Matrix is readily available it is possible to directly use this Matrix. This `PWM` should contain four rows (one for each base pair; ACTG in order). The number c olumns will depend on the length of the preferred binding motif of a given Transcription Factor. This argument is only necessary IF and ONLY IF, no `PFM` (Position Frequency Matrix) is available. Choosing between `PWM` or `PFM` comes down to personal choice as long a `PWM` is available for further computation (see `PFM`). If a `PFM` is available (see below), the Position Weight Matrix will be directly computed from the Position Frequency Matrix. Although it is possible to assign a new PWM to the `genomicProfileParameters` object without creating a new object, we suggest that if you were to decided to use another Position Weight Matrix to create a new `genomicProfileParameters`.

```
#Accessing PositionWeightMatrix slot
PositionWeightMatrix(GPP)
```

```
##          [,1]       [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## A -0.09520642 -1.0929970 -4.170092  1.761696  1.761696 -5.263560 -9.445015
## C  0.55082162  0.8819112 -4.550984 -9.445015 -9.445015 -9.445015  2.258075
## G  0.63156095 -2.0457025 -9.445015 -4.333846 -4.333846 -3.164873 -9.445015
## T -1.57041086  0.5565425  1.743852 -9.445015 -9.445015  1.735331 -9.445015
##        [,8]
## A -4.451342
## C  2.091309
## G -3.573736
## T -1.875062
```

```
# Setting PositionWeightMatrix slot
PositionWeightMatrix(GPP) <- newPWM
### This is not the advised method
### newPWM is a matrix following the format described above
```

- `PFM` , a Position Frequency Matrix. The Position Frequency Matrix argument may come in multiple forms: in the form of a Matrix containing four rows (one for each base pair ACTG) and columns depending of the length of the binding motif or in the form of a path to file linking to a `PFM`. Position Frequency Matricies come in various configurations. The most common ones (all supported by ChIPAnalyser) are RAW (similar to the simple matrix described previously), Transfac and JASPAR. Finally, if the

binding sequences are available, the PFM will be generated from sequence information. We suggest to use a path/to/file linking towards the PFM file. Most PFM will come in one of the formats described above and ChIPanalyser will parse these files in a usable format. However, PLEASE NOTE THAT THE FORMAT SHOULD BE SPECIFIED. See PFMFormat bellow.

If a PWM is readily available, PFM is not necessary. However, keep in mind that at least one is necessary. Although it is possible to assign a new PFM to the genomicProfileParameters object without creating a new object, we suggest that if you were to decided to use another Position Frequency Matrix to create a new genomicProfileParameters.

```
# Accessing PositionFrequencyMatrix slot
PositionFrequencyMatrix(GPP)
```

```
##   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## A  190   95   11  689  689    5    0    9
## C  213  268    6    0    0    0  696  620
## G  225   35    0    7    7   16    0   12
## T   68  298  679    0    0  675    0   55
```

```
# Setting PositionFrequencyMatrix slot
PositionFrequencyMatrix(GPP) <- newPFM
```

In this situation, newPFM is either a path to file or a PFM matrix. The PFMFormat will be the one assigned to the genomicProfileParameters object.

At least one of **PWM** or **PFM** is required to create a genomicProfileParameters storage object. If a PFM is provided then the PWM will be automatically computed and updated.

- PFMFormat, a file format for PositionFrequencyMatrix file. When Loading a PFM from a file (as described above), one should included the format of the file that they are using. PFMFormat may be one of the following: "raw","transfac","JASPAR" or "sequences".

```
PFMFormat(GPP)

PFMFormat(GPP)<-"raw"
```

Default is set at "raw".

All other arguments are optional however we strongly recommend to tailor the values assigned to genomicProfileParameters to your needs. The following sections will describe these optional parameters.

Genomic Parameters - The optional ones

- ScalingFactorPWM , a scaling factor for TF specificity. Although this parameter is optional (Default value is set at 1), the *scaling factor* (or *lambda* as described in the equations above) is crucial for many functions (described below). ScalingFactorPWM, must be a positive numeric value or a vector containing positive numeric values. The optimal value for ScalingFactorPWM may be inferred by using computeOptimal. Different values for ScalingFactorPWM will influence the goodness of fit of the model. For more information, see computeOptimal and profileAccuracyEstimate.

```
ScalingFactorPWM(GPP)

ScalingFactorPWM(GPP) <- 0.5

ScalingFactorPWM(GPP) <- c(0.5, 1, 1.5, 2)
```

- PWMpseudocount, a probability modifier. When computing a PWM from a PFM, it is possible that certain base pairs are completely absent from the Position Frequency Matrix. This absence will lead to odd

results as part of this transformation requires a logarithmic transformation (at Position probability matrix step - a Matrix that describes the simple probability of a base pair being in that position of a binding motif given the PFM). *zeroes* will give minus infinities. In order to overcome this problem,a `PWMpseudocount` is introduced in the Position Probability Matrix. a `PWMpseudocount` of 1 (Default Value is 1) will then become a 0 after logarithmic transformation thus removing any mathematical discomforts.

```
PWMpseudocount(GPP)

PWMpseudocount(GPP) <- 1
```

- `BPFrequency`, the frequency at which each base pair will occur in a given organism. Probabilistically speaking, all base pairs have an equal chance of occurring in the genome (Default value for this slot is set at 0.25 per base pair). However, biologically speaking this is not the case. `BPFrequency` may be supplied in various forms. If base pair frequency is known, it may be supplied as a vector containing the probability of occurrence of each base pair. If however, this frequency is unknown, `genomicProfileParameters` will compute `BPFrequency` from a `BSgenome` or a `DNAStringSet`. Bare in mind that `BPFrequency` is used to generate a PWM from a PFM, thus if one were to change the BPFrequency after creating a `genomicProfileParameters` with an already computed PWM , this would not influence the value of the PWM. It would be necessary to rebuild a new `genomicProfileParameters` object.

```
BPFrequency(GPP)

BPFrequency(GPP)<-c(0.2900342,0.2101426,0.2099192,0.2899039)

BPFrequency(GPP) <- DNASequenceSet
```

- `naturalLog`, a logical value. As described previously (see `pseudocount`), the transformation from PFM to PWM requires a logarithmic transformation. The user may choose which logarithmic transformation, they would rather apply (Default is `TRUE`). If `naturalLog = TRUE`, then the natural logarithm will be used for transformation. If `naturalLog = FALSE`, then *log2* will be used instead. Keep in mind that, the goal is to avoid any funky business during PFM to PWM transformation (e.g. Minus infinities or division by zero).

```
naturalLog(GPP)

naturalLog(GPP) <- FALSE
```

- `noOfSites` , the number of sites used to compute the `PWM` from the `PFM`. In the event that a `PFM` contains a large amount of sites (as it sometimes is the case with Transfac PFM), it is possible to restrict this number of sites. The default value is 0. When `noOfSites = 0`, the whole `PFM` is used to compute the `PWM`.

```
noOfSites(GPP)

noOfSites(GPP) <- 8
```

- `PWMThreshold`, a numeric threshold against which PWMScores are selected (Default is 0.7). Although it is possible to compute every single motif present in a stretch of DNA (if this is of interest, set `PWMThreshold` to 0), in most cases, only the sites with a high PWM Score will be of interest. The `PWMThreshold` , a numeric value between 0 and 1, will select regions above that given threshold. For the default threshold of 0.7, only the top 30% of PWMScores will be selected.

```
PWMThreshold(GPP)

PWMThreshold(GPP) <- 0.7
```

- `strandRule`, indicates how the genome should be scored with the `PWM` (Default is "max"). As DNA

is double stranded, it is necessary to specify how a strand of DNA should be scored. If `strandRule = "max"`, both strands will be scored and the highest score between each strand will be selected. If `strandRule = "sum"`, both strands will be scored and their respective score will be summed. If `strandRule = "mean"`, both strands will be score and the average score between both strands will selected as PWM Score. Only three possibilities: "max", "sum" and "mean"

```
strandRule(GPP)

strandRule(GPP) <- "mean"
```

- `whichstrand`, indicates which strand will be used to score the genome with the `PWM` (Default is both strand and is indicated by "+-"). Three options exist: plus strand ("+"), minus strand ("-") or both ("+-" or "-+").

```
whichstrand(GPP)

whichstrand(GPP) <- "+"
```

Genomic Parameters - The Updated ones

Some of the slots `genomicProfileParameters` should not be changed by user. We strongly advise against changing these slots. Certain Parameters are updated after a certain computation has been carried out. For example, `maxPWMScore` and `minPWMScore` are computed during the `computeGenomeWidePWMScore` function (see below) and represent both the highest and the lowest score of the given DNA sequence. These slots will be updated in the `genomicProfileParameters` object as one makes its way through the ChIPAnalyser work flow. Essentially, they are place holders for information required further down the work flow. Only slots that are of interest for the user are available for visualisation. If these slots have note been updated, the function will not return any value.

- `maxPWMScore`, a numeric value describing the highest PWM Score on a given DNA sequence and the value assigned to `lambda`. It is still possible to access this slot using:

```
maxPWMScore(Occupancy)
```

```
## [1] 12.86543
```

- `minPWMScore`, a numeric value describing the lowest PWM Score on a given DNA sequence and the value assigned to `lambda`. It is possible to access this slot using:

```
minPWMScore(Occupancy)
```

```
## [1] -49.22865
```

- `averageExpPWMScore` a numeric value representing the exponential of the average PWM Score. This score depends on the values assigned to `lambda`. It is possible to access this slot using:

```
averageExpPWMScore(Occupancy)
```

```
## [1] 0.8457538
```

- `DNASequenceLength` , a numeric value describing the length of the DNA sequence used. Although theoretically one could provide this information, DNA length is automatically computed and the slot updated during `computeGenomeWidePWMScore` function. The length of this sequence is the length of the sequence used to compute the scores previously mentioned (maxPWMScore, minPWMScore and averageExpPWMScore). This means that if DNA accessibility data is provided, the length of the sequence will only be the length of the accessible DNA.

```
DNASequenceLength(Occupancy)
```

```
## [1] 3145351
```

- `NoAccess`, indicates if certain Loci of interest (see setSequence below) **do not** contain any accessible DNA. It is possible that certain of the loci you have chosen do not contain any accessible DNA (no overlap with DNA accessibility data provided). If this is the case, you will be notified during the computation and the loci will be s tored in the `NoAccess` slot.

```
NoAccess(Occupancy)
```

```
## [1] "-"
```

- `AllSitesAboveThreshold`, stores all sites above threshold with the associated PWM Score and Occupancy. This slot may contain a variety of objects however they all represent the same thing: it will always contain at its core a GRanges object (slot class defined as "GRlist" - can be one of the following `GRangesList` or `list`). This GRanges inlcudes sites above threshold (start, end and strand), PWMScores for those sites and possibly Occupancy (depending on what has already been computed). GRanges are encapsulated in a GRangesList as each GRanges represent a specific Loci. This GRangesList may also be encapsulated in a list. This list will represent a combination of `lambda` and number of bound Molecules (see `boundMolecules`). For more information on this list see `computeOccupancy`. It is possible to access this slot by using:

```
AllSitesAboveThreshold(Occupancy)
```

```
## $`lambda = 1.5 & boundMolecules = 1000`
## GRangesList object of length 1:
## $eve
## GRanges object with 420 ranges and 3 metadata columns:
##       seqnames          ranges strand |          PWMScore DNAaffinity
##          <Rle>       <IRanges>  <Rle> |         <numeric>   <numeric>
##   eve    chr2R 5860705-5860712      + | -1.84573024098586           1
##   eve    chr2R 5860709-5860716      + | -4.96148500199546           1
##   eve    chr2R 5860715-5860722      + |  8.81832070896316           1
##   eve    chr2R 5860728-5860735      + |  4.24981127739825           1
##   eve    chr2R 5860758-5860765      + | -5.25856937621247           1
##   ...      ...             ...    ... .                 ...         ...
##   eve    chr2R 5876629-5876636      + |  5.76325435176529           1
##   eve    chr2R 5876635-5876642      + | 0.824810948340001           1
##   eve    chr2R 5876641-5876648      - |  -5.0584607351313           1
##   eve    chr2R 5876666-5876673      + |  1.87745682827728           1
##   eve    chr2R 5876684-5876691      + | -2.38839005613713           1
##               Occupancy
##               <numeric>
##   eve 0.0915185584072193
##   eve 0.0914749225700571
##   eve  0.148659402751388
##   eve 0.0943624008602243
##   eve 0.0914737995560999
##   ...                ...
##   eve  0.099361641959952
##   eve 0.0917645162456286
##   eve 0.0914745312764083
##   eve 0.0920652829761032
##   eve 0.0915034155828825
##
## -------
## seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

```
# Or

searchSites(Occupancy)
```

```
## $`lambda = 1.5 & boundMolecules = 1000`
## GRangesList object of length 1:
## $eve
## GRanges object with 420 ranges and 3 metadata columns:
##       seqnames          ranges strand |          PWMScore DNAaffinity
##          <Rle>       <IRanges>  <Rle> |         <numeric>   <numeric>
##   eve    chr2R 5860705-5860712      + | -1.84573024098586           1
##   eve    chr2R 5860709-5860716      + | -4.96148500199546           1
##   eve    chr2R 5860715-5860722      + |  8.81832070896316           1
##   eve    chr2R 5860728-5860735      + |  4.24981127739825           1
##   eve    chr2R 5860758-5860765      + | -5.25856937621247           1
##   ...      ...             ...    ... .                ...         ...
##   eve    chr2R 5876629-5876636      + |  5.76325435176529           1
##   eve    chr2R 5876635-5876642      + | 0.824810948340001           1
##   eve    chr2R 5876641-5876648      - |  -5.0584607351313           1
##   eve    chr2R 5876666-5876673      + |  1.87745682827728           1
##   eve    chr2R 5876684-5876691      + | -2.38839005613713           1
##               Occupancy
##               <numeric>
##   eve 0.0915185584072193
##   eve 0.0914749225700571
##   eve  0.148659402751388
##   eve 0.0943624008602243
##   eve 0.0914737995560999
##   ...                ...
##   eve  0.099361641959952
##   eve 0.0917645162456286
##   eve 0.0914745312764083
##   eve 0.0920652829761032
##   eve 0.0915034155828825
##
## -------
## seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

The size of the `AllSitesAboveThreshold` slot will increase drastically as the number of values assinged to `ScalingFactorPWM` (or `lambda`) and `boundMolecules` increases. In order to navigate and search this slot with ease, it is possible to use the `searchSites` function (**See below: searchSites**).

Data Objects - Occupancy Profile Parameters

`genomicProfileParameters` represents a good chunk of the parameters needed to go through the entire ChIPAnalyser work flow. However, there are more to come! A second parameter storing object was created to handle non-compulsory parameters. This lightens `genomicProfileParameters` by handling part of the parameters.

This second S4 object is called `occupancyProfileParameters`. The interesting aspect of this object is that none of the slots are compulsory. This means that if not provided , a new `occupancyProfileParameters` object will be created internally. All default values will be used for further computation.

As stated previously, we strongly advise using custom parameters in order to increase goodness of

fit of model. The `processingChIPseq` function returns a list, the second element of which is an `occupancyProfileParameter` object. This specific object will contain both max signal and background directly extracted from you ChIP data. These slots are explained below.

However this function can also take a `occupancyProfileParameters` object as an argument if you wish to change the smoothing parameters (see `chipSd`,`chipMean` and `chipSmooth`). The `processingChIPseq` function will update the one that is provided.

** See processcingChIPseq below **

```
OPP <- occupancyProfileParameters(ploidy = 2 ,boundMolecules = 1000 ,
    backgroundSignal = 0 ,maxSignal = 1, chipMean = 150 , chipSd = 150 ,
    chipSmooth = 250 , stepSize = 10 ,
    removeBackground = 0 )
```

As it is the case with `genomicProfileParameters`, it is also possible to *access/set* each slot individually after having created an `occupancyProfileParameters` object. Each slot is described as the following:

- `ploidy`, the ploidy level of the of the organism of interest (Default is set at 2). This only considers simple polyploidy (or haploidy). The model does not (yet) consider hybrids such as wheat.

```
ploidy(OPP)
ploidy(OPP) <- 2
```

- `boundMolecules`, a positive integer (or vector of positive integers) describing the number of bound molecules (Transcription factors) to DNA (Default value is set at 2000). In this model, occupancy is reliant on the number of bound molecules. The number of molecules will influence the goodness of fit of the model. It is possible to infer the number of bound Molecules by using the `computeOptimal` function. For more information, see `computeOptimal` and `profileAccuracyEstimate`.

```
boundMolecules(OPP)
boundMolecules(OPP) <- 5000
```

- `backgroundSignal`, a numeric value representing the background Signal in real ChIP-seq data (Default is set at 0). The background siganl is defined as the mean ChIP score over the entire genome (see `processingChIPseq` below).

```
backgroundSignal(OPP)

backgroundSignal(OPP) <- 0.02550997
```

- `maxSignal`, a numeric value representing the maximum signal in real ChIP-seq data (Default is set at 1) See `processingChIPseq` below)

```
maxSignal(OPP)

maxSignal(OPP) <- 1.86
```

- `chipMean`, a numeric value representing the average peak width in base pairs in real ChIP-seq data (Default is set at 150).

```
chipMean(OPP)

chipMean(OPP) <- 150
```

- `chipSd`, a numeric value representing the standard deviation of peak width in real ChIP-seq data (Default is set at 150).

```
chipSd(OPP)
```

```
chipSd(OPP) <- 150
```

- `chipSmooth`, a numeric value representing the size of the window used for smoothing the profile (Default is set at 250). The goal of ChIPAnalyser is to produce ChIP-seq like profile from predicted high occupancy sites. In order to mimic these ChIP-seq profile, a smoothing algorithm is used to smooth occupancy profiles. This algorithm uses ChIP-seq parameters such as `chipMean`, `chipSd`, `maxSignal`, `backgroundSignal` and `chipSmooth`.

```
chipSmooth(OPP)

chipSmooth(OPP) <- 250
```

- `stepSize`, a numeric value describing the bin size (in base pairs) used for computing ChIP-seq like profiles (Default is set at 10). In the case of long sequences, it not always necessary to include ChIP-like occupancy at every base pair (mainly for speed and memory usage). `stepSize` will determine the size of the bins used to split your sequence of interest. As an example, if your sequence is 16 000 bp long with a `stepSize` of 10, the resulting profile will be composed of 1600 occupancy points.

```
stepSize(OPP)

stepSize(OPP) <- 10
```

- `removeBackground`, a numeric value describing a threshold at which Occupancy signals must be removed (Default is set at 0).

```
removeBackground(OPP)

removeBackground(OPP) <- 0
```

Work Flow - Analysis

Once a `genomicProfileParameter` object has been established, the rest of the analysis becomes fairly straight forward. Unless, you already have prior knowledge on the number of bound molecules (`boundMolecules`) and the PWM scaling factor (`ScalingFactorPWM` or referred to as *lambda*), we advise you to first infer the optimal set of parameters as described in `computeOptimal`. However, as this function is essentially a combination of all other functions in the package (with a little bit more magic to it), we will overview a simple analysis work flow first and finish with `computeOptimal` function and its associated plotting function `plotOptimalHeatMaps`.

ChIP Score extraction.

It can be required to provied ChIP data to some of the functions in ChIPanalyser. We provided a function that will extract and normalise ChIP data from various formats. Based on this score, an `occupancyProfileParameters` will also be builtor updated (see above) with value for the relevant slots (See `backgroundSignal` and `maxSignal`).

```
processingChIPseq(profile,loci=NULL,reduce=NULL,
    occupancyProfileParameters=NULL,noiseFilter="zero",peaks=NULL,
    Access=NULL,cores=1)
```

As input, this functions takes:

- `profile` a path to file containing ChIP scores (*bed,wig,bigBed,...*), a `GRanges` or a `data.frame`containing ChIP scores.

- `loci` argument is a `GRanges` of the loci of interest. If none are supplied, a set of Sequence will be built internally based on the different chromosome available in ChIP data.
- `reduce` is a numeric value describing how many regions are to be selected. If a large amount of regions are supplied (let's say you split your favourite genome into bins of 20kbp and built a `GRanges` with those bins), you may choose to select the top regions based on the average ChIP singal in that region or if `peaks` are supplied select the regions overlapping with `peaks` with the highest ChIP score.
- `occupancyProfileParameters` is an `occupancyProfileParameters` object. If you wish to change smoothing parameters (see `chipSd`,`chipMean` and `chipSmooth`) please provide the aformentioned object. `processingChIPseq` will return a updated `occupancyProfileParameters`.
- `peaks` is a path to file or `GRanges` containing ChIP peaks. This argument is generally used if reduce is **not NULL**
- `Access` is a `GRanges` object containing accesible DNA for a given organism/cell type. If reduce is not NULL, only regions with accessible DNA will be selected. If `peaks` is not NULL, regions with peaks *and* accessible DNA will be selected.
- `noiseFilter` is one of the following: zero, mean, median or sigmoid. The noise filter is the method by which noise is buffered. Zero will remove all score below 0. Mean and median will remove all score below the mean and median respectively. Finally, sigmoid will assign a weight to each score based on a logistic regression curve (midpoint = ChIP-seq score 95 quantile). All scores above the midpoint will receive a weight between 1 and 2. All score below will receive a score between 0 and 1.
- `cores` is the number of course that should be used to process ChIP data.

Genome Wide Scoring

In order to score the entire genome (or the accessible genome), it is possible to use the `computeGenomeWidePWMScore` function. The output of this function will be influenced by the value assigned to `lambda`. If more than one value was assigned to the scaling factor, parameters dependant on lambda will be updated accordingly (computed for each value of lambda). It is possible to run this functions and make use of multiple cores in order to decrease computational time. The arguments of the function are the following :

```
computeGenomeWidePWMScore(DNASequenceSet, genomicProfileParameters,
    DNAAccessibility = NULL,  GenomeWide = TRUE,cores=1, verbose = TRUE)
```

Input Data - Genome Wide scoring

As input, `computeGenomeWidePWMScore` requires to obligatory arguments: `DNASequenceSet` and `genomicProfileParameters`. `DNASequenceSet` comes in the form of the following:

```
DNASequenceSet
```

```
##   A DNAStringSet instance of length 15
##         width seq                                          names
##  [1] 23011544 CGACAATGCACGACAGAGG...ATGAACCCCCCTTTCAAA chr2L
##  [2] 21146708 GACCCGCTAGGAGATGTTG...TTTGCATTCTAGGAATTC chr2R
##  [3] 24543557 TAGGGAGAAATATGATCGC...AACCAAGTTAATGTTCGG chr3L
##  [4] 27905053 GAATTCTCTCTTGTTGTAG...TTCGCATTCTAGGAATTC chr3R
##  [5]  1351857 GAATTCGCGTCCGCTTACC...CGATTTGAGATATATGAA chr4
##  ...      ... ...
## [11]  2555491 AACGAGGCCCATTTCATAC...ATGCCATTCGCTAGAAGT chr3LHet
## [12]  2517507 CCCTGTTTGCATCAGCGTT...TAAAAACAATTTGCTCCC chr3RHet
## [13]   204112 TAGATAGATAGATAGATAG...ATCGGAGTTAATGTTTGC chrXHet
```

```
## [14]   347038 AGGGTCACGTAATGCTGAT...TTGTTTTCCCCGGGATTG chrYHet
## [15] 29004656 ATTGAAAATGGATTGCATT...CAAGACCTTTCAAGACAA chrUextra
```

DNASequenceSet may also come in the form of a BSgenome object. However, we advise to use a DNAStringSet for a question of ease and speed. If you are unfamiliar with BSgenome and DNAStringSet, the following example demonstrates how to use these objects in this context.

```
#Extracting DNAStringSet from BSgenome

DNASequenceSet <- getSeq(BSgenome.Dmelanogaster.UCSC.dm3)
```

As a reminder a genomicProfileParameters are presented in the following format:

```
GPP
```

```
## Object Class:genomicProfileParameters
##

##
## PWM:

##          [,1]       [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## A -0.09520642 -1.0929970 -4.170092  1.761696  1.761696 -5.263560 -9.445015
## C  0.55082162  0.8819112 -4.550984 -9.445015 -9.445015 -9.445015  2.258075
## G  0.63156095 -2.0457025 -9.445015 -4.333846 -4.333846 -3.164873 -9.445015
## T -1.57041086  0.5565425  1.743852 -9.445015 -9.445015  1.735331 -9.445015
##        [,8]
## A -4.451342
## C  2.091309
## G -3.573736
## T -1.875062

##
## PFM:

##   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## A  190   95   11  689  689    5    0    9
## C  213  268    6    0    0    0  696  620
## G  225   35    0    7    7   16    0   12
## T   68  298  679    0    0  675    0   55

##
## PFMFormat: raw

##
## PWM Scores at Sites higher than Threshold:

## Warning in showList(object, showFunction, print.classinfo = TRUE): Note that starting with BioC 3.7,
##   GRangesList **instances** needs to be set to
##   "CompressedGRangesList". Please update this object with
##   'updateObject(object, verbose=TRUE)' and re-serialize it.

## GRangesList object of length 0:
## <0 elements>
##
## -------
## seqinfo: no sequences

##
## No Accessible DNA at Loci:
```

```
##
##
## Genomic Profile Parameters:
## Lambda:  1
## BP Frequency:    0.2916399   0.2088135   0.2085611   0.2909855
## Pseudocount: 1
## Natural log: FALSE
## Number Of Sites: 0
## maxPWMScore:
## minPWMScore:
## PWMThreshold: 0.7
## Average Exponential PWM Score:
## DNA Sequence Length:
## Strand Rule: max
## Strand: +-
```

`DNAAccessibility` is an optional argument in `computeGenomeWidePWMScore`. If present, then the genome will be scored only on the accessible DNA. `DNAAccessibility` comes as a `GRanges` containing accessible DNA sites.

```
# DNA accessibility
Access
```

```
## GRanges object with 4703 ranges and 0 metadata columns:
##          seqnames            ranges strand
##             <Rle>         <IRanges>  <Rle>
##      [1]    chr2R   7339296-7342564      *
##      [2]    chr2R   9436993-9437589      *
##      [3]    chr2R 15728083-15728687      *
##      [4]    chr2R   4980200-4980845      *
##      [5]    chr2R   6028863-6029419      *
##      ...      ...               ...    ...
##   [4699]    chr2R 21120053-21120400      *
##   [4700]    chr2R 21140572-21140980      *
##   [4701]    chr2R 21143160-21143517      *
##   [4702]    chr2R 21144932-21145281      *
##   [4703]    chr2R 21145564-21146702      *
##   -------
##   seqinfo: 6 sequences from an unspecified genome; no seqlengths
```

`verbose` will determine if progress messages should be printed in the console and cores will determine the number of cores that will be used to compute genome wide metrics.

computeGenomeWidePWMScore

As an example of `computeGenomeWidePWMScore` usage:

```
# With DNAAccessibility

GenomeWide <- computeGenomeWidePWMScore(DNASequenceSet = DNASequenceSet,
    genomicProfileParameters = GPP, DNAAccessibility = Access,cores=1)

GenomeWide
```

```
# Without DNA accessibility

GenomeWide <- computeGenomeWidePWMScore(DNASequenceSet = DNASequenceSet,
    genomicProfileParameters = GPP,cores=1)
GenomeWide
```

Scoring sites above threshold

Once genome wide metrics have been computed, the next step in the analysis is to extract sites above threshold (Sites with strong binding sites according to PWM Scores). The `computePWMScore` function will score the genome and extract sites above a local threshold (dependant on `PWMThreshold`, `maxPWMScore` and `minPWMScore`). It is possible to run this functions and make use of multiple cores in order to decrease computational time. The arguments of this functions are the following:

```
computePWMScore(DNASequenceSet, genomicProfileParameter,
    setSequence = NULL, DNAAccessibility = NULL, cores=1 ,verbose = TRUE)
```

Input Data - Sites Above threshold

Only two arguments are absolutely required: `DNASequenceSet` and `genomicProfileParameters`. However, `setSequence` represents the Loci of interest. If `setSequence = NULL`, then sites above threshold will computed and extracted on a genome wide scale (or accessible genome if DNA Accessibility is provided). `DNASequenceSet` and `DNAAccessibility` are in the same format as previously described (`verbose` plays the same role as previously described). `setSequence` is a `GRanges` representing the loci of interest (may contain more than one loci/range) and comes in the following format:

```
eveLocus
```

```
## GRanges object with 1 range and 0 metadata columns:
##       seqnames          ranges strand
##          <Rle>       <IRanges>  <Rle>
##   eve    chr2R 5860693-5876692      *
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

An important aspect to mention, is that it is recommended you name your loci of interest (not to be confused with `seqnames`). If no names are supplied they will be named internally following the format:

- ChromosomeName_startOfRange..endOfRange

If you are unfamiliar with GRanges, the following examples demonstrates naming in the context of ChIP-Analyser. We recommend getting acquainted with GenomicRanges as many aspect of ChIPanalyser require the use of GRanges.

```
# Sequence names of Loci
seqnames(eveLocus)
```

```
## factor-Rle of length 1 with 1 run
##   Lengths:     1
##   Values : chr2R
## Levels(1): chr2R
```

```
# Names of Loci

names(eveLocus)
```

```
## [1] "eve"
```

```
# Naming Loci in GRanges
names(eveLocus) <- "eve"
```

computePWMScore

To compute PWM Scores at sites above threshold:

```
# With DNA Accessibility

PWMScores <- computePWMScore(DNASequenceSet = DNASequenceSet,
    genomicProfileParameters = GenomeWide,
    setSequence = eveLocus, DNAAccessibility = Access,cores=1)
PWMScores

# Without DNA Accessibility

PWMScores <- computePWMScore(DNASequenceSet = DNASequenceSet,
    genomicProfileParameters = GenomeWide,
    setSequence = eveLocus,cores=1)
PWMScores
```

As you can see, the `genomicProfileParameters` argument is the `genomicProfileParameters` object computed in the previous example. ChIPAnalyser works in a sequential manner: resulting objects from one function are often parsed as arguments to other functions. Finally, if your sequence of interest does not contain any accessible DNA, you will be notified during the computation and it is possible to extract inaccessible loci by using `NoAccess(PWMScores)` (See NoAccess slot in `genomicProfileParameters`).

Occupancy

Occupancy scores are computed using the formula described in **Methods**. It is worth mentioning that Occupancy scores are dependant on values assigned to `ScalingFactorPWM` and `boundMolecules`. If more than one value were to be assigned to these parameters, the resulting output will be a combination of both. For more information see the `computeOptimal` example as we will demonstrate multiple value computation (Single Value for lambda and boundMolecules will return an object identical in structure as with multiple values). The arguments for `computeOccupancy` are the following:

```
computeOccupancy(AllSitesPWMScore, occupancyProfileParameters = NULL,
    norm = TRUE,verbose = TRUE)
```

Input Data - Occupancy

`computeOccupancy` requires a `genomicProfileParameters` object result of the previous function (`computePWMScore`). If you are unsure, if your `genomicProfileParameter` contains the right information, it is possible to check by using:

```
AllSitesAboveThreshold(PWMScores)
```

If your GRanges does not contain PWMScore as a metadata column, you are either using the wrong object or you have not yet computed PWM Scores.

`occupancyProfileParameters` is an `occupancyProfileParameters` object. If not provided, a new one will be generated internally. As previously mentioned, we strongly recommend to set those parameters to improve

the model's goodness of it. As a reminder, a `occupancyProfileParameters` object (previously created - see section **Data object - Occupancy profile Parameters**) should print on the screen as follows:

```
OPP
```

```
## Object Class:occupancyProfileParameters
##
```

```
## Ploidy: 2
```

```
## boundMolecules:  1000
```

```
## backgroundSignal: 0.0914686723280862
## maxSignal: 1
## chipMean: 150
## chipSd: 150
## chipSmooth: 250
## Step Size: 10
```

Finally, if `norm = TRUE`, the occupancy profiles will be normalised and `verbose = TRUE` progress messages will be printed to the console.

computeOccupancy

To compute Occupancy scores with `computeOccupancy`:

```
Occupancy <- computeOccupancy(AllSitesPWMScore = PWMScores,
    occupancyProfileParameters = OPP)
Occupancy
```

As it is the case in the previous functions, `AllSitesPWMScore` should be the result of the previous function (`computePWMScore`). `computeOccupancy` will return a `genomicProfileParameters` object with an updated `AllSitesAboveThreshold` slot. This slot should now contain a list of GRangesLists containing GRanges (one for each Loci of interest) with two metadata columns (PWMScore and Occupancy). Each element in the list is named with the specific combination of *lambda* and *boundMolecules* used to compute this set of occupancies. Finally, if your sequence of interest does not contain any accessible DNA, you will be notified during the computation and it is possible to extract inaccessible loci by using `NoAccess(PWMScores)` (See NoAccess slot in `genomicProfileParameters`).

ChIP-seq like profiles

The ultimate goal of ChIPAnalyser is to produce *ChIP-seq like* profile from occupancy data (from sites that display a high TF occupancy). `computeChipProfile` creates *ChIP-seq like* profiles from occupancy data by smoothing occupancy *profiles* and mimicking real ChIP-seq data. It is possible to run this functions and make use of multiple cores in order to decrease computational time. The arguments of `computeChipProfile` are the following:

```
computeChipProfile( setSequence ,
    occupancy, occupancyProfileParameters = NULL, norm = TRUE,
    method = c("moving_kernel","truncated_kernel","exact"),
    peakSignificantThreshold= NULL,cores=1
    verbose = TRUE)
```

Input data - ChIP-seq profiles

The `computeChipProfile` function requires two compulsory arguments `setSequence` and `occupancy`. `setSequence` is a GRanges describing the loci of interest (this is the same GRanges used in `computePWMScore`). `occupancy` is a `genomicProfileParameters` object result of `computeOccupancy` function. To make sure this is the right `genomicProfileParameters`, you may use `AllSitesAboveThreshold()` (See AllSitesAboveThreshold slot description above). `occupancyProfileParameters` is an `occupancyProfileParameters` object. If not supplied, it will be generated *de novo* internally. Once again, we recommend to set the parameters of this object in relationship to real ChIP-seq data. `norm = TRUE` and `method` respectively represent if the ChIP-seq like profile should be normalised and if you wish to use an approximation for ChIP-seq profile or not. `moving_kernel` will use Rcpp to approximate and compute peaks, `truncated_kernel` will also approximate peaks but without using Rcpp, and `exact` will not approximate peaks. These methods represent different way of computing and/or approximating ChIP-seq peaks. Finally, `peakSignificantThreshold` is a threshold at which peaks will be selected. If you select "moving_kernel" then this threshold is a numeric value describing the peak tail hight cut-off value. The default in this case is 0.001. In the case of "truncated_kernel" and "exact", the threshold represents a distance in base pair from the peak summit at which the peak should be cut. In this case, default is set at 1250 base pairs.

It should be noted that these methods will produce very similar results. And by very similar results, we mean nearly identical.

computeChipProfile

To generate a ChIP-seq like profile:

```
chipProfile <- computeChipProfile(setSequence = eveLocus,
    occupancy = Occupancy,occupancyProfileParameters = OPP,cores=1)
chipProfile
```

The output of this functions is slightly different as it returns a named list (each element in the list is named after the specific combination of *lambda* and *boundMolecules* used to compute occupancies) containing a GRangesList of GRanges with ChIP profile values as a metadata column. These GRanges also differ in the sense that they now contain the whole loci (or accessible loci) cut into bins of size equal to `stepSize` (See stepSize slot in `occupancyProfileParameters`). Each GRangesList contains GRanges for each Loci of interest.

Searching through SitesAboveThreshold and ChIP-seq profiles

As described previously, The size of the `AllSitesAboveThreshold` slot will increase drastically as the number of values assigned to `ScalingFactorPWM` (or `lambda`) and `boundMolecules` increases. In order to navigate and search this slot with ease, it is possible to use the `searchSites` function. This function may also be used on predicted ChIP-seq profiles (result of `computeChipProfile`). `searchSites` comes in the following form:

```
searchSites(Sites,ScalingFactor="all", BoundMolecules="all",Locus="all")
```

It is possible to use this function as a simple extraction method similarly to the `AllSitesAboveThreshold` method. In this case, the usage is the following:

```
searchSites(Occupancy)
```

```
## $`lambda = 1.5 & boundMolecules = 1000`
## GRangesList object of length 1:
## $eve
## GRanges object with 420 ranges and 3 metadata columns:
##       seqnames          ranges strand |          PWMScore DNAaffinity
```

```
##          <Rle>       <IRanges>  <Rle> |         <numeric>   <numeric>
##   eve    chr2R 5860705-5860712      + | -1.84573024098586           1
##   eve    chr2R 5860709-5860716      + | -4.96148500199546           1
##   eve    chr2R 5860715-5860722      + |  8.81832070896316           1
##   eve    chr2R 5860728-5860735      + |  4.24981127739825           1
##   eve    chr2R 5860758-5860765      + | -5.25856937621247           1
##   ...      ...             ...    ... .                ...         ...
##   eve    chr2R 5876629-5876636      + |  5.76325435176529           1
##   eve    chr2R 5876635-5876642      + | 0.824810948340001           1
##   eve    chr2R 5876641-5876648      - |  -5.0584607351313           1
##   eve    chr2R 5876666-5876673      + |  1.87745682827728           1
##   eve    chr2R 5876684-5876691      + | -2.38839005613713           1
##               Occupancy
##               <numeric>
##   eve 0.0915185584072193
##   eve 0.0914749225700571
##   eve  0.148659402751388
##   eve 0.0943624008602243
##   eve 0.0914737995560999
##   ...                ...
##   eve  0.099361641959952
##   eve 0.0917645162456286
##   eve 0.0914745312764083
##   eve 0.0920652829761032
##   eve 0.0915034155828825
##
## -------
## seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

If you wish to navigate and extract only certain combinations of `ScalingFactorPWM` and/or `boundMolecules` and/or Loci, `searchSites` could be use as shown below:

```
searchSites(chipProfile, ScalingFactor=c(1.5,2.5), BoundMolecules=c(1000,1500)
    ,Locus=c("eve","odd"))
```

```
## $`lambda = 1.5 & boundMolecules = 1000`
## $`lambda = 1.5 & boundMolecules = 1000`$eve
## GRanges object with 1600 ranges and 1 metadata column:
##       seqnames          ranges strand |              ChIP
##          <Rle>       <IRanges>  <Rle> |         <numeric>
##   eve    chr2R 5860693-5860703      * | 0.0633101738995645
##   eve    chr2R 5860703-5860713      * | 0.0686501364849457
##   eve    chr2R 5860713-5860723      * | 0.0741342776797116
##   eve    chr2R 5860723-5860733      * | 0.0797869804734581
##   eve    chr2R 5860733-5860743      * | 0.0856333772959523
##   ...      ...             ...    ... .                ...
##   eve    chr2R 5876643-5876653      * | 0.0800444269473348
##   eve    chr2R 5876653-5876663      * | 0.0762385945900301
##   eve    chr2R 5876663-5876673      * | 0.0723418265102848
##   eve    chr2R 5876673-5876683      * | 0.0683367973234721
##   eve    chr2R 5876683-5876693      * | 0.0642057003062555
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

Estimating the accuracy of the model

In order to determine how accurate the predicted model is, it is possible to compare the predicted *ChIP-seq like profile* (as built in `computeChipProfile`) to real ChIP-seq data for a given Transcription Factors at loci of interest. `profileAccuracyEstimate` provides a way to compare both profiles. The arguments for this function are the following:

```
profileAccuracyEstimate(LocusProfile,
    predictedProfile, occupancyProfileParameters = NULL,method="all")
```

Input data - Accuracy Estimate

`profileAccuracyEstimate` requires only two arguments. `precitedProfile` is the result of `computeChipProfile`. Finally, `LocusProfile` is a list containing actual ChIP-seq profiles. This list is the result of the `processingChIPseq` function.

We also strongly recommend that each loci in `LocusProfile` (each element of the list) should be named in an identical manner as the loci used in `setSequence` (See previous functions). This list should come in the following format:

```
str(eveLocusChip)
```

```
## List of 1
##  $ eve: num [1:16000] 0.0108 0.0108 0.0108 0.0108 0.0108 ...
```

In this example, there is only one element in the list. However, this list can be as long as you wish and contain all the Loci that you are interested in.

`occupancyProfileParameters` is only required if you have change the step size slot. Each predicted profile, will only contain a fraction of the score (see stepSize)

Finally, `method` is the quality assessment method that you wish to use. The following possibilities are available: pearson, spearman, kendall, ks, geometric,fscore and all. `pearson,spearman and kendall` are correlation methods. `ks` is a Kolmogorov-Smirnov test (Will return both KS Distance and KS p-value) `geometric` is in an house metric that describes the ratio of difference in the area between curves over shared area between curves. `fscore` is a combination of multiple metrics such as Precision,recall,Fscore,Accuracy,MCC, and AUC. `all` is a combination of all the above mentioned.

In every case, the Mean Squared Error will also be returned.

profileAccuracyEstimate

To test the accuracy the model against ChIP-seq data:

```
AccuracyEstimate <- profileAccuracyEstimate(LocusProfile = eveLocusChip,
    predictedProfile = chipProfile, occupancyProfileParameters = OPP)
AccuracyEstimate
```

This function returns a list of two elements. The first element represents lists containing the model quality assessments for every combination of parameters (Bound Molecules and lambda) for every genomic region. The second element of the list contains the result of the ROCR package: False positives, False Negative, etc...

Generally Speaking, we recommend to only use the first element of this list. However, we offer the possibility to choose from the other metrics available in the ROCR package by returning the ROCR prediction object.

Finding optimal Parameters

As described previously, it is not always possible to know the optimal set of parameters for `ScalingFactorPWM` and `boundMolecules`. `ChIPAnalyser` offers the possibility to backward infer the parameters using the `computeOptimal` function. By testing different combinations of `ScalingFactorPWM` and `boundMolecules`, the `computeOptimal` function will return a list of values that were the highest ranking parameter combination for each model quality assessment method selected (see`profileAccuracyEstimate` above). This function will also return a list of matricies containing relevant scores for each combination. It is possible to run this functions and make use of multiple cores in order to decrease computational time. Values that should be tested for `ScalingFactorPWM` and for `boundMolecules` should be provided by user. If these values are not provided (default value and only one value for each parameter), then they will be assigned internally. The internal values are the following:

```
ScalingFactorPWM(genomicProfileParameters) <- c(0.25, 0.5, 0.75, 1, 1.25,
        1.5, 1.75, 2, 2.5, 3, 3.5 ,4 ,4.5, 5)

boundMolecules(occupancyProfileParameters) <- c(1, 10, 20, 50, 100,
        200, 500,1000,2000, 5000,10000,20000,50000, 100000,
        200000, 500000, 1000000)
```

In terms of its arguments,`computeOptimal` can be described as:

```
computeOptimal(DNASequenceSet,
    genomicProfileParameters,
    LocusProfile,
    setSequence,
    DNAAccessibility = NULL,
    occupancyProfileParameters = NULL,
    optimalMethod = "all",
    peakMethod="moving_kernel"
    cores=1)
```

Please note that this functions will take some time to complete. Do not be alarmed if it seems to have stalled.

Input Data - Optimal Parameters

`computeOptimal` is essentially a combination of previous functions (with a bit more magic to it). For this reason, data input in extremely similar to the functions described above. As a quick reminder:

- `DNASequenceSet`, a DNAStringSet (or BSgenome) containing the sequences of the organism of interest.
- `genomicProfileParameters`, a `genomicProfileParameters` object containing at least a *Position Weight Matrix* or *Position Frequency Matrix*. All other slots will be computed internally.
- `LocusProfile`, a named list of ChIP-seq profile for loci of interest.
- `setSequence`, a named GRanges containing loci of interest.
- `DNAAccessibility`, a GRanges containing Accessible DNA.
- `occupancyProfileParameters`, an `occupancyProfileParameters` object. Although optional, we strongly advise to tailor this object by using values directly extracted from `LocusProfile`

`optimalMethod` defines which metric you wish to compute. There are four possible choices: *pearson*, *spearman*, *kendall*, *ks*, *geometric*, *fscore* or *all* It is imperative that the lists/GRanges are named with the name of the Loci of interest. `peakMethod`describes if you wish to use an approximation for ChIP-seq profile peaks. `moving_kernel` will use Rcpp to approximate and compute peaks, `truncated_kernel` will also approximate peaks but without using Rcpp, and `exact` will not approximate peaks. These methods represent different way of computing and/or approximating ChIP-seq peaks.

Finally, cores describes the number of cores that will be used to compute the optimal set of parameters.

computeOptimal

As a example describing the usage of `compute optimal`

```
optimalParam <- computeOptimal(DNASequenceSet = DNASequenceSet,
    genomicProfileParameters = GPP,
    LocusProfile = eveLocusChip,
    setSequence = eveLocus,
    DNAAccessibility = Access,
    occupancyProfileParameters = OPP,
    optimalMethod = "all",
    cores=1)
optimalParam
```

The `computeOptimal` function will return a list of values that were the highest ranking parameter combination for each model quality assessment method selected (see`profileAccuracyEstimate` above). This function will also return a list of matricies containing relevant scores for each combination.

Plotting Results

As it is the case in mamy fields, data visualisation is a key aspect in any analysis. For this purpose, `ChIPAnalyser` offers two plotting functions: `plotOptimalHeatMaps` and `plotOccupancyProfile`.

Optimal Parameters

Once you have computed the optimal set of parameters, it is possible to plot these results in the form of a heat map using `plotOptimalHeatMaps`. Depending on what you are interested in, this function will either plot *correlation ,MSE, theta* or *all of the previous*. This functions requires minimal input as described below:

```
plotOptimalHeatMaps(optimalParam=optimalParam,contour=TRUE,col=NULL,main=NULL,
    layout=TRUE)
```

Input Data & Plotting

`plotOptimalHeatMaps` only requires one data input in the form of the result of `computeOptimal` (see `computeOptimal`). `contour` defines if contour lines should be plotted. `col` are the colors that you wish to use for your heatmaps. It should be noted that the color vector will be recycled if not enough colors are provided. `main` is the main title. Finally, `layout` is a logical value that defines if standard layout should be selected. Standard layout will generate an individual heatmap for each matrix provided (`computeOptimal` result) with a heat map scale bar on the left.

NOTE: If you use your own layout, you should be aware that the `plotOptimalHeatMaps` will always plot both the heat map and the scale bar. In R this is considered as two plots (rasterImage scales).

As an example:

```
plotOptimalHeatMaps(optimalParam)
```

See plot in **Quick Guide**

The boxed tile represents the highest correlation or theta for a given combination of `ScalingFactorPWM` and `boundMolecules`. In the case of MSE the boxed tile represents the lowest Mean Squared Error.

Plotting Profiles

`ChIPAnalyser` produces ChIP-seq like profiles. It is possible to plot these profiles but also to add a variety of features to these plots as well graphical parameter parsing. `plotOccupancyProfile` takes care of plotting with the following arguments:

```
plotOccupancyProfile <- function(predictedProfile,
    setSequence,
    chipProfile = NULL,
    DNAAccessibility = NULL,
    occupancy = NULL,
    profileAccuracy=NULL,
    PWM=FALSE,
    occupancyProfileParameters = NULL,
    geneRef = NULL,axis=TRUE,...)
```

Input Data & Profiles

In order to increase plotting flexibility, `plotOccupancyProfile` only plots one profile at a time. In practice, this means that only simple data units should be parsed to this functions. This also means that the main title is left to the user discretion. The arguments described above should come in the following format:

- `precitedProfile`, a GRanges object containing the predicted ChIP-seq like profile for *one* locus and one combination of `lambda` and `boundMolecules`.
- `setSequence`, a GRanges object containing the locus of interest.
- `profileAccuracy`, the profile Accuracy estimate for one loci and for *one* combination of `lambda` and `boundMolecules`
- `chipProfile`, a vector containing ChIP-seq data for locus of interest. In previous functions, ChIP-seq data was stored in a named list. In this case, it is the individual numeric vector contained within that list.
- `occupancy`, a GRanges object containing both PWMScore and Occupancy. This GRanges is the result of `computeOccupancy` and should only contain a GRanges object for one locus and one combination of `lambda` and `boundMolecules`.
- `PWM`, a logical operator indicating wherever you wish to plot *occupancy* or *PWMScores*. It is necessary to also include `occupancy` data.
- `DNAAccessibility`, a GRanges object containing DNAAccessibility. DNAAccessibility is similar to DNAAccessibility data described previously.
- `occupancyProfileParameters`, an `occupancyProfileParameters` object. This object should be the same as the one used in functions described above. However, the minimal requirement is that the `stepSize` slot remains consistent with `stepSize` used previously. As a reminder, stepSize default value is set at 10.
- `geneRef`, a List containing genetic information (3'UTR, 5'UTR, exons, intron and enhancers). Each element of this list, is a GRanges containing the information regarding 3'UTR, 5'UTR, exons, intron and enhancers.
- `axis` determine if the axes should be included
- `...` Any other graphical Parameter of the following : col, density, border, lty, lwd, cex, cex.axis, xlab, ylab, xlim, ylim, las and axislables.

As this object has not yet be described, `geneRef` should come in a similar format as the following:

```
geneRef
```

```
## GRanges object with 7 ranges and 2 metadata columns:
##       seqnames          ranges strand |            type          ID
##          <Rle>       <IRanges>  <Rle> |     <character> <character>
```

```
##   [1]     chr2R 5866746-5867058      + |            exon         eve
##   [2]     chr2R 5866746-5866919      + |  five_prime_UTR         eve
##   [3]     chr2R 5867059-5867129      + |          intron         eve
##   [4]     chr2R 5867130-5868284      + |            exon         eve
##   [5]     chr2R 5868122-5868284      + | three_prime_UTR         eve
##   [6]     chr2R 5876666-5876808      + |            exon       TER94
##   [7]     chr2R 5876666-5876791      + |  five_prime_UTR       TER94
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

It should be noted that only two arguments are necessary (`predictedProfile` and `setSequence`). The more arguments are provided the more information will be plotted. As an example:

```
plotOccupancyProfile(predictedProfile=chipProfile[[1]][[1]],
    setSequence=eveLocus,
    chipProfile = eveLocusChip[[1]],
    DNAAccessibility = Access,
    occupancy = AllSitesAboveThreshold(Occupancy)[[1]][[1]],
    occupancyProfileParameters = OPP,
    geneRef =geneRef)
```

Session Information

```
sessionInfo()
```

```
## R version 3.6.0 (2019-04-26)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows Server 2012 R2 x64 (build 9600)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=C
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] BSgenome.Dmelanogaster.UCSC.dm3_1.4.0
##  [2] ChIPanalyser_1.6.0
##  [3] RcppRoll_0.3.0
##  [4] BSgenome_1.52.0
##  [5] rtracklayer_1.44.0
##  [6] Biostrings_2.52.0
##  [7] XVector_0.24.0
##  [8] GenomicRanges_1.36.0
##  [9] GenomeInfoDb_1.20.0
## [10] IRanges_2.18.0
```

```
## [11] S4Vectors_0.22.0
## [12] BiocGenerics_0.30.0
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.1                  compiler_3.6.0
##  [3] BiocManager_1.30.4          bitops_1.0-6
##  [5] tools_3.6.0                 zlibbioc_1.30.0
##  [7] digest_0.6.18               evaluate_0.13
##  [9] lattice_0.20-38             Matrix_1.2-17
## [11] DelayedArray_0.10.0         yaml_2.2.0
## [13] xfun_0.6                    GenomeInfoDbData_1.2.1
## [15] stringr_1.4.0               knitr_1.22
## [17] caTools_1.17.1.2            gtools_3.8.1
## [19] grid_3.6.0                  Biobase_2.44.0
## [21] XML_3.98-1.19               BiocParallel_1.18.0
## [23] rmarkdown_1.12              gdata_2.18.0
## [25] ROCR_1.0-7                  magrittr_1.5
## [27] Rsamtools_2.0.0             gplots_3.0.1.1
## [29] htmltools_0.3.6             matrixStats_0.54.0
## [31] GenomicAlignments_1.20.0    SummarizedExperiment_1.14.0
## [33] KernSmooth_2.23-15          stringi_1.4.3
## [35] RCurl_1.95-4.12
```

References

Zabet NR, Adryan B (2015) Estimating binding properties of transcription factors from genome-wide binding profiles. Nucleic Acids Res., 43, 84–94.