The ChIPanalyser User's Guide

Patrick Martin

Introduction - What is this package?

ChIPanalyser provides a quick an easy method to predict and explain TF binding. The package uses a statistical thermodynamic framework to model the bidning of proteins to DNA. ChIPanalyser makes the assumption that the probability that any given sites along the genome is bound by a TF can be explained by four main factors: DNA accessibility, Binding energy, a scaling factor modulating binding energy (λ) and number of TF bound (N) to DNA. Both N and λ are inferred internally by maximisng (or minimising) a goodness of fit metric between predictions and actual ChIP-data. The package will produce a ChIP like profile at a base pair level. As opposed to machine learning type frameworks, if these paramters are known by other means (experimentally), ChIPanalyser does require any training to produce ChIP like profiles. Estimated values can directly be plugged into the model. Furthermore, ChIPanalyser helps gain an understanding of the mechanims behind TF binding as described by (Zabet et al 2015 & Martin and Zabet 2019).

Methods - The Model

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

The next section will be split between the following subsections

- **Loading Data** - Description of internal Data. We will use this data for our work flow example.
- **Quick Start** - We will give quick start example. Only core functionalities and work flow will be described in this section.
- **Advanced Work** - We will describe a more indepth work flow.
- **Parameter Description** - We will give an in depth description of each parameter used in ChIPAnalyser

Loading Data

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
# We recommend using the latest version of the genome
# Please ensure that all your data is aligned to the same version of the genome

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*in *RAW* format. we provide loading support for JASPAR, Transfac and RAW. If you wish to use any other format, we suggest to use the MotifDb package (or load PFM as matrix into R) and parse the matrix to ChIPanalyser. In this scenario, `PFMFormat` argument should be set to *matrix* (see below for more information).
- `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.

IMPORTANT: Data sets provided in the package have been currated to meet the size requirements for Bioconductor packages. Please read the instruction below carefully as we will describe how to incorportate your own data into the pipe line.

Quick Start

Step 1 - Extracting Normalised ChIP scores from ChIP-seq datasets

ChIPAnalyser requires ChIP data in order to infer the optimal set of values that will be assigned to bound Molecules and lambda. The package will maximise (or minimise) a goodness of fit metric between the predictions and ChIP data.

If you have inferred or approximated the values to be assigned to N and λ by other means, skip this step and go directly to **Advanced Work**.

`eveLocusChip` can be a connection to your ChIP data, a GRanges of your ChIP or data frame (*bed* format style) `loci` is a GRanges object containing loci of interest. Default set as NULL (see **Advanced Work**)

```
eveLocusChip<-processingChIP(profile=eveLocusChip,
                             loci=eveLocus,
                             cores=1)
eveLocusChip
```

```
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##                        ChIP score from 1 regions
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
## [ 1 ]  eve
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##
##                        Top  1  regions
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
## 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
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##
##                        Associated Options
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##  backgroundSignal:                     0.0981378770503508
##  maxSignal:                                             1
##  chipMean:                                            200
##  chipSd:                                              200
##  chipSmooth:                                          250
##  lociWidth:                                         20000
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

The output of `processingChIP` returns a `ChIPScore` object containing extracted and normalised ChIP scores, your loci of interest and internal parameters associated to the ChIP extractions process.

Step 2 - Computing a PWMs

The model relies on a Postion Weight Matrix. `genomicProfiles` serve as a way of storing important paramters. More importantly, this object stores intermediate results as you make your way through the analysis pipeline.

From a Position Frequency Matrix:

```
# PFMs are automatically converted to PWM when build genomicProfiles
GP<-genomicProfiles(PFM=PFM,PFMFormat="raw")
GP
```

```
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##
##                         genomicProfiles object: Position Weight Matrix
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##
## Position Weight Matrix (PWM):
##
##          [,1]       [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## A  0.08784795 -0.6039860 -2.740123  1.375113  1.375113 -3.502263 -6.546785
## C  0.20197414  0.4314283 -3.327910 -6.546785 -6.546785 -6.546785  1.385218
## G  0.25671985 -1.5980255 -6.546785 -3.179490 -3.179490 -2.372398 -6.546785
## T -0.93731362  0.5374410  1.360498 -6.546785 -6.546785  1.354592 -6.546785
##        [,8]
## A -2.935867
## C  1.269632
## G -2.654965
## T -1.148623
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##
## Base pairs frequency for PWM weighting :
##
##    A    C    G    T
## 0.25 0.25 0.25 0.25
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##
## PWM built from (PFM - Format  raw ) :
##
##   [,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
##
##  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

From a Position Weight Matrix:

```
GP<-genomicProfiles(PWM=PositionWeightMatrix)
```

Step 3 - Computing Optimal Parameters

As described above, ChIPanalyser infers the optimal combination of bound molecules and lambda that will maximise (or minimise) a goodness of fit metric. The following function computes the optimal set of parameters and returns intermediate steps of the analysis pipeline. To do so, we will be parsing the follwing:

a DNA sequence, a Position Weight Matrix (contained in a genomicProfiles), chromatin States (Accessible DNA - This is optional), and extracted/normalised experimental ChIP score (result of the `processingChIP` function).

```
## surpress dependency warnings
optimal<-suppressWarnings(computeOptimal(genomicProfiles=GP,
                         DNASequenceSet=DNASequenceSet,
                         ChIPScore=eveLocusChip,
                         chromatinState=Access))
```

```
## Computing Genome Wide PWM Score

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

## PWM Scores Extraction

## Computing Occupancy

## Computing ChIP-seq-like Profile

## Computing Accuracy of Profile
```

NOTE Default Optimal parameters are provided internally as the following:

```
 ## Lambda Values
 seq(0.25,5,by=0.25)
```

```
##  [1] 0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00 2.25 2.50 2.75 3.00 3.25 3.50
## [15] 3.75 4.00 4.25 4.50 4.75 5.00
```

```
 ## Bound Molecule Values
 c(1, 10, 20, 50, 100,
    200, 500,1000,2000, 5000,10000,20000,50000, 100000,
    200000, 500000, 1000000)
```

```
##  [1] 1e+00 1e+01 2e+01 5e+01 1e+02 2e+02 5e+02 1e+03 2e+03 5e+03 1e+04
## [12] 2e+04 5e+04 1e+05 2e+05 5e+05 1e+06
```

NOTE To change these parameters see **Advanced Work**

Step 4 - Extracting Optimal Paramters (Prelim)

Once computed, we will extract the optimal set of parameters.

```
optimalParam<-optimal$Optimal
optimalParam$OptimalParameters
```

```
## $pearson
##         lambda boundMolecules
##          5e-01          1e+06
##
## $spearman
##         lambda boundMolecules
##           1.25         200.00
##
## $kendall
##         lambda boundMolecules
##        7.5e-01        1.0e+05
##
## $KsDist
```

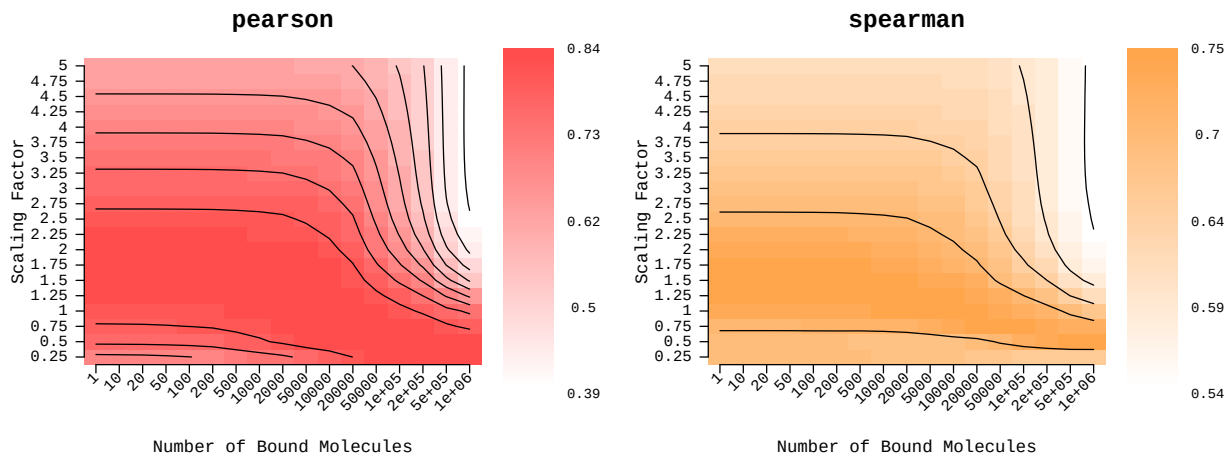
```
##         lambda boundMolecules
##              2           5000
##
## $geometric
##         lambda boundMolecules
##           1.25       10000.00
##
## $precision
##         lambda boundMolecules
##        2.5e-01        1.0e+05
##
## $recall
##         lambda boundMolecules
##           1.25           1.00
##
## $f1
##         lambda boundMolecules
##           1.25          10.00
##
## $accuracy
##         lambda boundMolecules
##           0.25           1.00
##
## $MCC
##         lambda boundMolecules
##           1.25          10.00
##
## $AUC
##         lambda boundMolecules
##           1.25           1.00
##
## $MSE
##         lambda boundMolecules
##           1.25       10000.00
```

We can see the opitmal set of parameters suggested by ChIPanalyser.

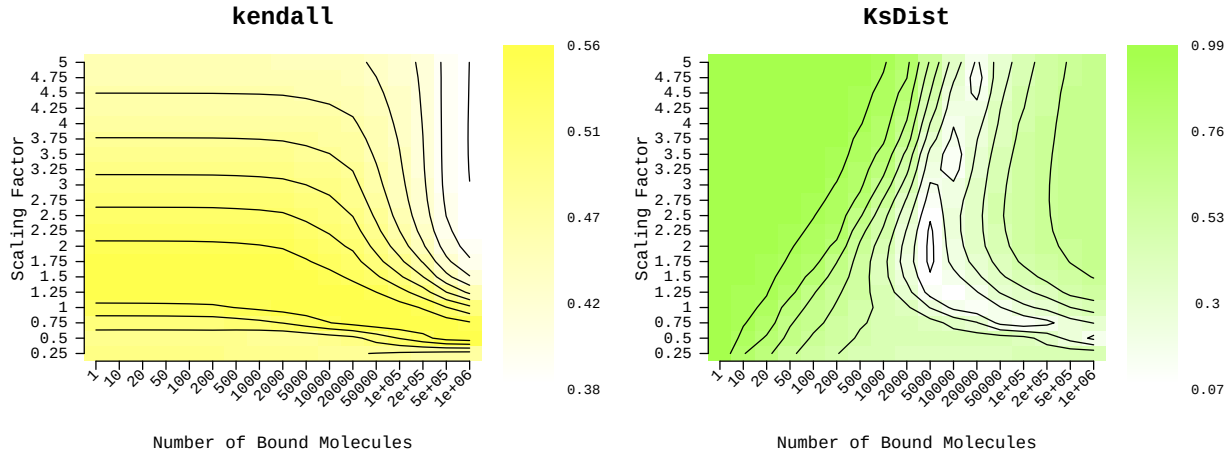
Step 5 - Plotting Optimal Set of Parameters

Despite ChIPanalyser returning suggested optimal parameters, you may wish to visualise the optimal parameters for each paramter combination and choose your own set of parameters. To this effect, we have implemented an Optimal parameter plotting fucntion.

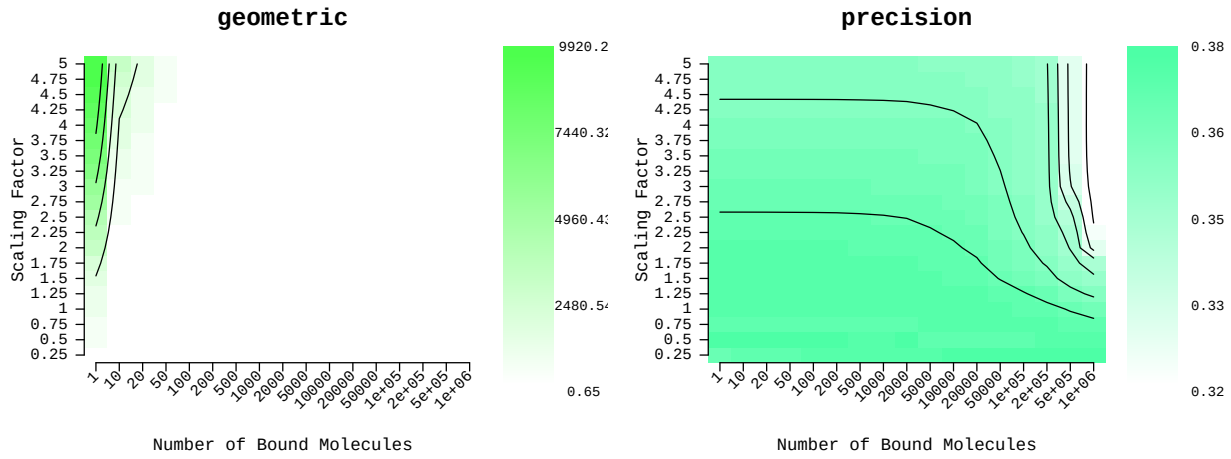
```
# Plotting Optimal heat maps
par(oma=c(0,0,3,0))
layout(matrix(1:8,ncol=4, byrow=T),width=c(6,1.5,6,1.5),height=c(1,1))
plotOptimalHeatMaps(optimalParam,layout=FALSE)
```


pearson
Scaling Factor
Number of Bound Molecules
0.84
0.73
0.62
0.5
0.39

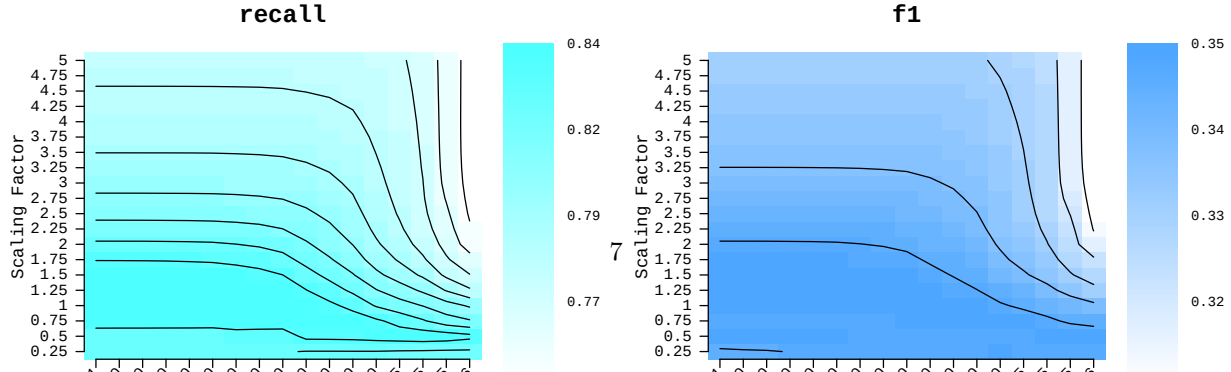
spearman
Scaling Factor
Number of Bound Molecules
0.75
0.7
0.64
0.59
0.54


kendall
Scaling Factor
Number of Bound Molecules
0.56
0.51
0.47
0.42
0.38

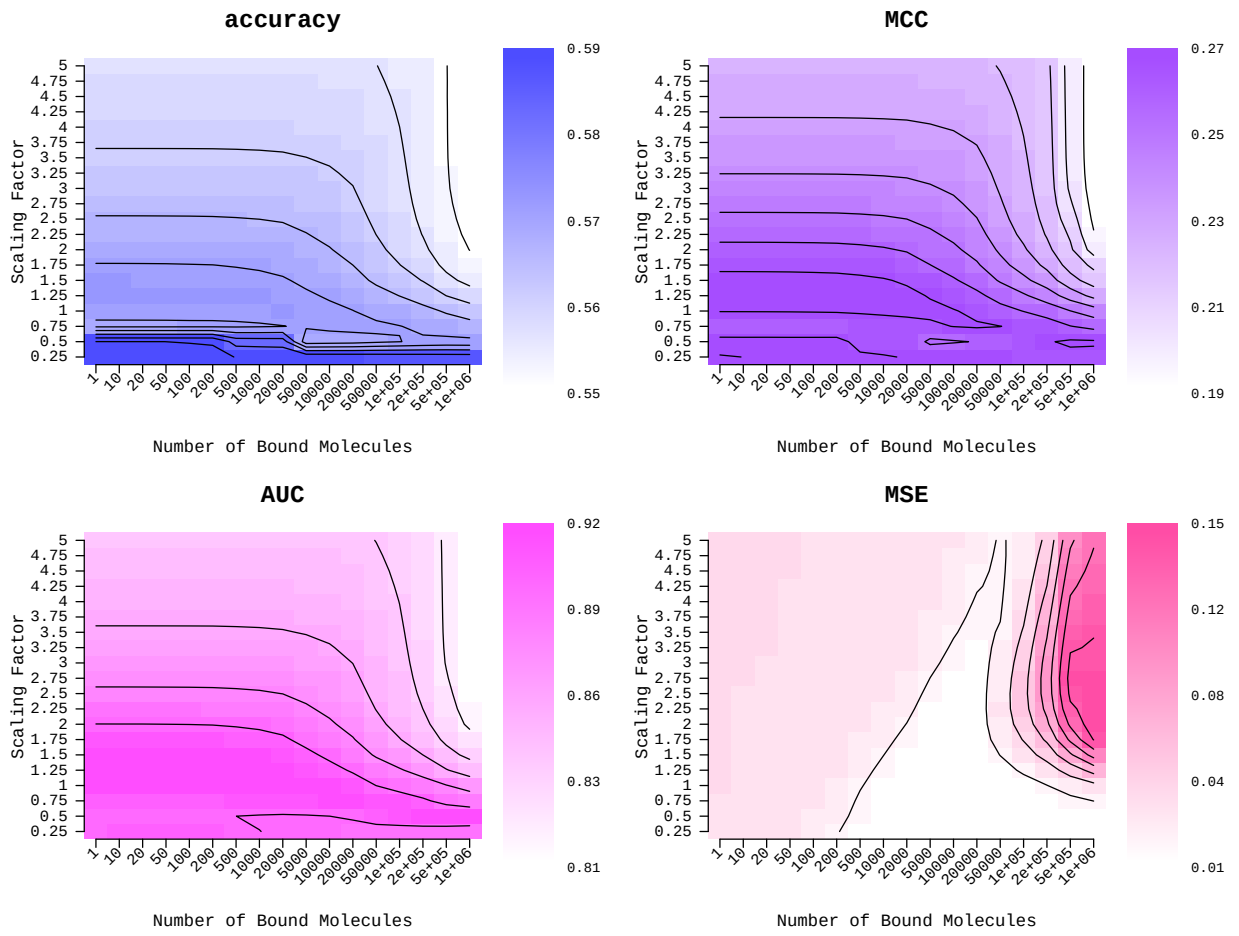
KsDist
Scaling Factor
Number of Bound Molecules
0.99
0.76
0.53
0.3
0.07


geometric
Scaling Factor
Number of Bound Molecules
9920.2
7440.3
4960.4
2480.5
0.65

precision
Scaling Factor
Number of Bound Molecules
0.38
0.36
0.35
0.33
0.32


recall
Scaling Factor
0.84
0.82
0.79
0.77

f1
Scaling Factor
0.35
0.34
0.33
0.32



Step 6 - Extracting Optimal Set of Parameters with associated data

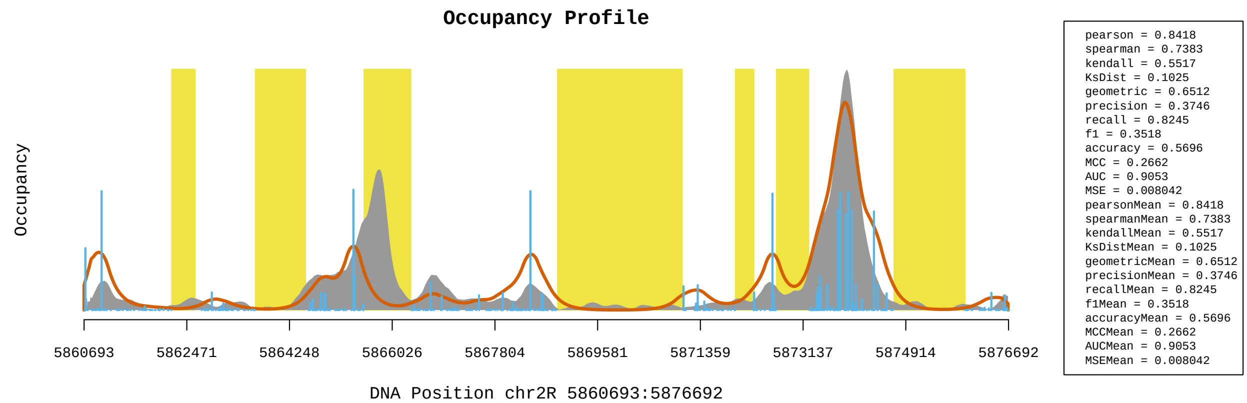
Once satisfied with your choice of optimal parameters, you can extracted the data associated to those parameters. You can select more that one parameter however the values assigned to each argument must be one of the values used for computing the optimal set of parameters.

```
optimalParam<-searchSites(optimal,lambdaPWM=1.25,BoundMolecules=10000)
```

Step 7 - Plotting ChIP_seq like profiles

The final step is to plot the computed predicted profiles. We provide a plotting function to produce predicted profile plots.

```
plotOccupancyProfile(predictedProfile=optimalParam$ChIPProfiles,
                     ChIPScore=eveLocusChip,
                     chromatinState=Access,
                     occupancy=optimalParam$Occupancy,
                     goodnessOfFit=optimalParam$goodnessOfFit)
```



Advanced Work

In this section we will describe a more in depth work flow. This will include parameter tweakin, annexe functions and some insights into the outputs of each functions.

Step 1 - Parameter Set Up

Some parameters can be changed between each step of the pipeline. These parameters will enable you to tweak and improve the quality of your analysis.

There are many parameters to choose from. These parameters already have default value assigned to them.

```
## Suggested Parameters to start with.
parameterOptions()
```

```
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##                        parameterOptions
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##  processingChIP options
##
##  chipMean:                                        200
##  chipSd:                                          200
##  chipSmooth:                                      250
##  lociWidth:                                     20000
##  noiseFilter:                                    zero
##
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##  processingChIP options Updated
##
##  maxSignal:                                         1
##  backgroundSignal:                                  0
##
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##  Position Weight Matrix Options
##
##  naturalLog:                                     TRUE
##  noOfSites:                                       all
##  PWMpseudocount:                                    1
```

```
##
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##  Genome Wide Score options
##
##  strandRule:                                            max
##  whichstrand:                                            +-
##  lambdaPWM:                                               1
##
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##  PWM Scores above Threshold options
##
##  strandRule:                                            max
##  whichstrand:                                            +-
##
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##  Occupancy options
##
##  Ploidy:                                                  2
##  lambdaPWM:                                               1
##  boundMolecules:                                       1000
##  maxSignal:                                               1
##  backgroundSignal:                                        0
##
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##  ChIP Profile options
##
##  chipMean:                                              200
##  chipSd:                                                200
##  chipSmooth:                                            250
##  stepSize:                                               10
##
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

```
## Changing parameters
PO <- parameterOptions(noiseFilter="sigmoid",chipSd=150,chipMean=150,lociWidth=30000)
```

NOTE If you wish to do so, you can change all your parameters at this step. These parameters will be parsed through the different steps of the pipeline as long as you parse this object to each step of the analysis.

Step 2 - Extracting Normalised ChIP scores.

In some case you will not have a pre-determined idea of which loci you wish to look at. The `processingChIP` function offers a few possibilities to work around this issue.

```
## Top 50 loci based on ChIP score
processingChIP(profile="/path/to/ChIP",
               loci=NULL,
               reduce=50,
               parameterOptions=PO)

## Top 50 loci ALSO containing peaks
processingChIP(profile="/path/to/ChIP",
               loci=NULL,
               reduce=50,
               peaks="/path/to/peaks",
```

```
                parameterOptions=PO)

## Top 50 loci containing BOTH peaks and Accessible DNA
processingChIP(profile="/path/to/ChIP",
                loci=NULL,
                reduce=50,
                peaks="/path/to/peaks",
                chromatinState="/path/to/chromatinState"
                parameterOptions=PO)
```

Loci will be computed internally based on the ChIP score provided. ChIP Scores will be tilled into bins of width equals to the value assigned to the `lociWidth` argument in the `parameterOptions` object (see above). The default loci width is set at 20 kbp. Top regions are selected based on ordered ChIP scores.

Most genomic formats are supported by ChIPanalyser (wig, bed, bedGraph, bigWig, bigBed). The "path/to/file" may also be a GRanges object.

`processingChIP` function returns extracted/Normalised ChIP scores (list of numeric vectors), the loci of inetrest (GRanges), and associated paramters that have been extracted (such as maxSignal and backgroundSignal). The loci are the top n regions as selected by the reduce argument. Using the `loci()` accessor applied on a ChIPScore object will return a GRanges of selected loci. The `scores()` accessors applied on a `ChIPScore` object will return the normalised scores associated to each Locus.

NOTE This function also supports multi core computing.

Step 3 - Position Weight Matrix and Associated Paramters

Computing the PWM from PFMs can be tweaked by using some additional parameters. PWMs depend on Base Pair Frequency in the genome of interest. Either you can provide a vector containing the base pair frequency (A C T G in order) or the `genomicProfiles` object will compute it internally if you provide a BSgenome/DNAStringSet.

```
str(genomicProfiles())
```

```
## Formal class 'genomicProfiles' [package "ChIPanalyser"] with 31 slots
##   ..@ PWM               : num[0 , 0 ]
##   ..@ PFM               : num[0 , 0 ]
##   ..@ PFMFormat         : chr "raw"
##   ..@ BPFrequency       : num [1:4] 0.25 0.25 0.25 0.25
##   ..@ minPWMScore       : logi(0)
##   ..@ maxPWMScore       : logi(0)
##   ..@ profiles          :Formal class 'CompressedGRangesList' [package "GenomicRanges"] with 5 slots
##   .. .. ..@ unlistData     :Formal class 'GRanges' [package "GenomicRanges"] with 7 slots
##   .. .. .. .. ..@ seqnames       :Formal class 'Rle' [package "S4Vectors"] with 4 slots
##   .. .. .. .. .. .. ..@ values         : Factor w/ 0 levels:
##   .. .. .. .. .. .. ..@ lengths        : int(0)
##   .. .. .. .. .. .. ..@ elementMetadata: NULL
##   .. .. .. .. .. .. ..@ metadata       : list()
##   .. .. .. .. ..@ ranges         :Formal class 'IRanges' [package "IRanges"] with 6 slots
##   .. .. .. .. .. .. ..@ start          : int(0)
##   .. .. .. .. .. .. ..@ width          : int(0)
##   .. .. .. .. .. .. ..@ NAMES          : NULL
##   .. .. .. .. .. .. ..@ elementType    : chr "ANY"
##   .. .. .. .. .. .. ..@ elementMetadata: NULL
##   .. .. .. .. .. .. ..@ metadata       : list()
```

```
##   .. .. .. .. ..@ strand         :Formal class 'Rle' [package "S4Vectors"] with 4 slots
##   .. .. .. .. .. .. ..@ values         : Factor w/ 3 levels "+","-","*":
##   .. .. .. .. .. .. ..@ lengths        : int(0)
##   .. .. .. .. .. .. ..@ elementMetadata: NULL
##   .. .. .. .. .. .. ..@ metadata       : list()
##   .. .. .. .. ..@ seqinfo        :Formal class 'Seqinfo' [package "GenomeInfoDb"] with 4 slots
##   .. .. .. .. .. .. ..@ seqnames   : chr(0)
##   .. .. .. .. .. .. ..@ seqlengths : int(0)
##   .. .. .. .. .. .. ..@ is_circular: logi(0)
##   .. .. .. .. .. .. ..@ genome     : chr(0)
##   .. .. .. .. ..@ elementMetadata:Formal class 'DFrame' [package "S4Vectors"] with 6 slots
##   .. .. .. .. .. .. ..@ rownames       : NULL
##   .. .. .. .. .. .. ..@ nrows          : int 0
##   .. .. .. .. .. .. ..@ listData       : Named list()
##   .. .. .. .. .. .. ..@ elementType    : chr "ANY"
##   .. .. .. .. .. .. ..@ elementMetadata: NULL
##   .. .. .. .. .. .. ..@ metadata       : list()
##   .. .. .. .. ..@ elementType    : chr "ANY"
##   .. .. .. .. ..@ metadata       : list()
##   .. .. ..@ elementMetadata:Formal class 'DataFrame' [package "S4Vectors"] with 6 slots
##   .. .. .. .. ..@ rownames       : NULL
##   .. .. .. .. ..@ nrows          : int 0
##   .. .. .. .. ..@ listData       : Named list()
##   .. .. .. .. ..@ elementType    : chr "ANY"
##   .. .. .. .. ..@ elementMetadata: NULL
##   .. .. .. .. ..@ metadata       : list()
##   .. .. ..@ elementType    : chr "GRanges"
##   .. .. ..@ metadata       : list()
##   .. .. ..@ partitioning   :Formal class 'PartitioningByEnd' [package "IRanges"] with 5 slots
##   .. .. .. .. ..@ end            : int(0)
##   .. .. .. .. ..@ NAMES          : NULL
##   .. .. .. .. ..@ elementType    : chr "ANY"
##   .. .. .. .. ..@ elementMetadata: NULL
##   .. .. .. .. ..@ metadata       : list()
##   ..@ DNASequenceLength : logi(0)
##   ..@ averageExpPWMScore: logi(0)
##   ..@ ZeroBackground    : logi(0)
##   ..@ drop              : logi(0)
##   ..@ tags              : chr "empty"
##   ..@ ploidy            : num 2
##   ..@ boundMolecules    : num 1000
##   ..@ backgroundSignal  : num 0
##   ..@ maxSignal         : num 1
##   ..@ lociWidth         : num 20000
##   ..@ chipMean          : num 200
##   ..@ chipSd            : num 200
##   ..@ chipSmooth        : num 250
##   ..@ stepSize          : num 10
##   ..@ removeBackground  : num 0
##   ..@ noiseFilter       : chr "zero"
##   ..@ PWMThreshold      : num 0.7
##   ..@ strandRule        : chr "max"
##   ..@ whichstrand       : chr "+-"
##   ..@ lambdaPWM         : num 1
```

```
##   ..@ naturalLog        : logi TRUE
##   ..@ noOfSites         : chr "all"
##   ..@ PWMpseudocount    : num 1
##   ..@ paramTag          : chr "empty"
```

```
GP <- genomicProfiles(PFM=PFM, PFMFormat="raw", BPFrequency=DNASequenceSet)
GP
```

```
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##
##                         genomicProfiles object: Position Weight Matrix
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##
## Position Weight Matrix (PWM):
##
##          [,1]       [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## A -0.06599206 -0.7576078 -2.890487  1.221115  1.221115 -3.648422 -6.546785
## C  0.38180045  0.6112943 -3.154502 -6.546785 -6.546785 -6.546785  1.565178
## G  0.43776469 -1.4179730 -6.546785 -3.003993 -3.003993 -2.193723 -6.546785
## T -1.08852586  0.3857658  1.208746 -6.546785 -6.546785  1.202840 -6.546785
##        [,8]
## A -3.085435
## C  1.449585
## G -2.477125
## T -1.299694
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##
## Base pairs frequency for PWM weighting :
##
##         A         C         G         T
## 0.2916399 0.2088135 0.2085611 0.2909855
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##
## PWM built from (PFM - Format  raw ) :
##
##   [,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
##
##  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

The `genomicProfiles` object also contains all parameters described in `parameterOptions`. This makes the parsing of parameters straight forward between each step of the analysis pipeline. The `genomicProfiles` object will be updated internally as you provided More parameters. This object will also be the main object that you will parse through each step of the analysis. There are a few different ways to add new paramters:

```
## Parsing pre computed parameters (processingChIP function)
GP<-genomicProfiles(PFM=PFM, PFMFormat="raw", BPFrequency=DNASequenceSet,
                    ChIPScore=ChIPScore)

## Parsing pre assigned function (parameterOptions)
parameterOptions<-parameterOptions(lambdaPWM=c(1,2,3),
                                   boundMolecules=c(5,50,500))
GP<-genomicProfiles(PFM=PFM, PFMFormat="raw", BPFrequency=DNASequenceSet,
```

```
                    parameterOptions=parameterOptions)

## Direct parameter assignement

GP<-genomicProfiles(PFM=PFM, PFMFormat="raw", BPFrequency=DNASequenceSet,
                    lambdaPWM=c(1,2,3),boundMolecules=c(4,500,8000))
```

NOTE `parameterOptions` object can be parsed to any function of the analysis pipeline if parameters need to be changed along the way.

Step 4 - Computing Optimal Set of Parameters

The optimal set of parameters can be computed on custom set of values for N and λ. As described above, there a few ways to modify parameter Options. If you were to assign more than two values to both of these slots, these new values will be used as "Optimal Parameters Combinations". NOTE `parameterOptions` is inherited by `genomicProfiles` hence why you can also assign those paramters to the `genomicProfiles` contructor.

The `computeOptimal` function offers a few more options that we will briefly describe here.

```
## Setting custom parameters
OP<-parameterOptions(lambdaPWM=seq(1,10,by=0.5),
                     boundMolecules=seq(1,100000, length.out=20))

## Computing ONLY Optimal Parameters and MSE as goodness Of Fit metric
optimal<-computeOptimal(genomicProfiles=GP,
                        DNASequenceSet=DNASequenceSet,
                        ChIPScore=eveLocusChip,
                        chromatinState=Access,
                        parameterOptions=OP,
                        optimalMethod="MSE",
                        returnAll=FALSE)

### Computing ONLY Optimal Parameters and using Rank slection method
optimal<-computeOptimal(genomicProfiles=GP,
                        DNASequenceSet=DNASequenceSet,
                        ChIPScore=eveLocusChip,
                        chromatinState=Access,
                        parameterOptions=OP,
                        optimalMethod="all",
                        rank=TRUE)
```

When the `returnAll` argument is set to *FALSE*, only the optimal parameters Will be return. No internal Data will be returned.

Optimal Parameters are determined by selecting the best perfomring combination of paremeters. The goodness of fit score for each combination is averaged over all regions considered. When the `rank` argument is set to *TRUE*, the optimal parameters will be based on which combination of parameters showed the best performance for each regions individually. Parameter combinations are ranked based on how many individual regions performed best with that specific set of parameters.

Finally, `optimalMethod` argument will enbale you to selected the goodness Of Fit method you wish to use.

Step 5 - Extracting and Plotting Optimal Parameters

Now that you have selected custom parameters, you will want to plot the associated heat maps.

```
## Extracted Optimal Parameters
optimalParam<-optimal$Optimal

## Plotting heat maps
plotOptimalHeatMaps(optimalParam,overlay=TRUE)
```

It is possible to plot an overlay of the optimal set of paramters of all goodness Of Fit methods. Using the `overlay` argument in the plotting function will plot overlay the top 10% of optimal parameters as selected by each Goodness of fit metric.

Step 6 - Computing individual parameter combinations

Let's imagine that when looking at the optimal parameter heat maps, you would like to run a combination of parameters that is not in the ones that had been provided but you do not want to re-compute optimal parameters. Or Let us imagine that you have already an estimate of number of bound molecules. ChIPanalyser provides functions that will enable you to run the piple line on individual parameter combinations. The steps are described as following:

```
## Creating genomic Profiles object with PFMs and associated parameters
GP <- genomicProfiles(PFM=PFM,PFMFormat="raw",BPFrequency=DNASequenceSet,
                      lambdaPWM=1, boundMolecules=58794)

## Computing Genome Wide Score required
GW <- computeGenomeWideScores(genomicProfiles=GP,
                              DNASequenceSet=DNASequenceSet,
                              chromatinState=Access)
```

```
## Extracting genome wide scores

## Considering Chromatin State ~ Both strands

## Computing Mean waiting time
```

```
GW
```

```
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##
##                         genomicProfiles object: Accessible Chromatin Score
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##
##  Position Weight Matrix (PWM):
##
##          [,1]       [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## A -0.06599206 -0.7576078 -2.890487  1.221115  1.221115 -3.648422 -6.546785
## C  0.38180045  0.6112943 -3.154502 -6.546785 -6.546785 -6.546785  1.565178
## G  0.43776469 -1.4179730 -6.546785 -3.003993 -3.003993 -2.193723 -6.546785
## T -1.08852586  0.3857658  1.208746 -6.546785 -6.546785  1.202840 -6.546785
##        [,8]
## A -3.085435
## C  1.449585
## G -2.477125
## T -1.299694
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

```
##
##  maxPWMScore:                            8.91763676310073
##  minPWMScore:                           -34.1227030233141
##  averageExpPWMScore:         1.12386639831079 (lambda = 1)
##  DNASequenceLength:                              3145351
## - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
##
```

```
## Computing PWM score above threshold
pwm <- computePWMScore(genomicProfiles=GW,
                       DNASequenceSet=DNASequenceSet,
                       loci=eveLocusChip,
                       chromatinState=Access)
```

```
## PWM Scores Extraction
pwm
```

```
## - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
##
##                        genomicProfiles object: PWM scores above Threshold
## - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
##
##  Scores above Threshold in locus:
## $eve
## GRanges object with 420 ranges and 2 metadata columns:
##       seqnames          ranges strand |          PWMScore DNAaffinity
##          <Rle>       <IRanges>  <Rle> |         <numeric>   <numeric>
##   eve    chr2R 5860705-5860712      + | -1.27936271261358           1
##   eve    chr2R 5860709-5860716      + | -3.43903934052361           1
##   eve    chr2R 5860715-5860722      + |  6.11239413669119           1
##   eve    chr2R 5860728-5860735      + |  2.94574470484046           1
##   eve    chr2R 5860758-5860765      + | -3.64496253690054           1
##   ...      ...             ...    ... .                ...         ...
##   eve    chr2R 5876629-5876636      + |  3.99478350477594           1
##   eve    chr2R 5876635-5876642      + | 0.571715383336846           1
##   eve    chr2R 5876641-5876648      - | -3.50625779652945           1
##   eve    chr2R 5876666-5876673      + |  1.30135390714341           1
##   eve    chr2R 5876684-5876691      + | -1.65550583348886           1
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
##
## - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
##
##                        Associated parameters
## - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
##
##  minPWMScore:                           -34.1227030233141
##  maxPWMScore:                            8.91763676310073
##  lambdaPWM:                                             1
##  PWMThreshold:                                        0.7
```

```
## Computing Occupancy of sites above threshold

occup <- computeOccupancy(genomicProfiles=pwm)
```

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

```r
occup
```

```
## - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
##
##                        genomicProfiles object: Occupancy at sites above Threshold
## - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
##
##  Parameter combination:  lambda = 1 & boundMolecules = 58794
##
## $`lambda = 1 & boundMolecules = 58794`
## 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      + | -1.27936271261358           1
##   eve    chr2R 5860709-5860716      + | -3.43903934052361           1
##   eve    chr2R 5860715-5860722      + |  6.11239413669119           1
##   eve    chr2R 5860728-5860735      + |  2.94574470484046           1
##   eve    chr2R 5860758-5860765      + | -3.64496253690054           1
##   ...      ...             ...    ... .               ...         ...
##   eve    chr2R 5876629-5876636      + |  3.99478350477594           1
##   eve    chr2R 5876635-5876642      + | 0.571715383336846           1
##   eve    chr2R 5876641-5876648      - | -3.50625779652945           1
##   eve    chr2R 5876666-5876673      + |  1.30135390714341           1
##   eve    chr2R 5876684-5876691      + | -1.65550583348886           1
##                  Occupancy
##                  <numeric>
##   eve  0.00230831670412893
##   eve 0.000266837890397655
##   eve    0.789652344600473
##   eve     0.13660033245351
##   eve 0.000217188966182787
##   ...                  ...
##   eve    0.311143336824885
##   eve   0.0145164829977214
##   eve 0.000249495333839159
##   eve   0.0296495854390687
##   eve  0.00158581639171441
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
##
##
## - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
##
##                        Associated parameters
## - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
##
##  lambdaPWM:                                                 1
##  boundMolecules:                                        58794
```

```r
## Compute ChIP seq like profiles

chip <- computeChIPProfile(genomicProfiles=occup,
```

```r
                              loci=eveLocusChip)
```

```
## Computing ChIP Profile
```

```r
chip
```

```
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##
##                        genomicProfiles object: ChIP like Profiles
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##
##  Parameter combination:  lambda = 1 & boundMolecules = 58794
##
## $`lambda = 1 & boundMolecules = 58794`
## GRangesList object of length 1:
## $eve
## GRanges object with 1600 ranges and 1 metadata column:
##       seqnames          ranges strand |              ChIP
##          <Rle>       <IRanges>  <Rle> |         <numeric>
##   eve    chr2R 5860693-5860702      * | 0.254601657999258
##   eve    chr2R 5860703-5860712      * | 0.267655364144033
##   eve    chr2R 5860713-5860722      * | 0.281333841525761
##   eve    chr2R 5860723-5860732      * | 0.280479411740008
##   eve    chr2R 5860733-5860742      * | 0.274875112367782
##   ...      ...             ...    ... .               ...
##   eve    chr2R 5876643-5876652      * | 0.117035247267259
##   eve    chr2R 5876653-5876662      * | 0.111995569256909
##   eve    chr2R 5876663-5876672      * | 0.107235938505588
##   eve    chr2R 5876673-5876682      * | 0.102244271003216
##   eve    chr2R 5876683-5876692      * | 0.097293939248763
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
##
##
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##
##                        Associated parameter Options
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##
##  lambdaPWM:                                                    1
##  boundMolecules:                                           58794
##  stepSize:                                                    10
##  backgroundSignal:                            0.0981378770503508
##  maxSignal:                                                    1
##  removeBackground:                                             0
##  chipMean:                                                   200
##  chipSd:                                                     200
##  chipSmooth:                                                 250
```

```r
## Compute goodness Of Fit of model
accu <- profileAccuracyEstimate(genomicProfiles=chip,
                                ChIPScore=eveLocusChip)
```

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

```r
accu
```

```
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##
##                         genomicProfiles object: Goodness of Fit
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##
##  Selected Goodness of Fit metrics:
##  pearson
##  spearman
##  kendall
##  KsDist
##  geometric
##  precision
##  recall
##  f1
##  accuracy
##  MCC
##  AUC
##  MSE
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##
##  1 parameter combinations for 1 loci
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##
##  Parameter combination:  lambda = 1 & boundMolecules = 58794
##
##   pearsonMean  spearmanMean   kendallMean    KsDistMean geometricMean
##    0.81174664    0.72081285    0.54320733    0.24562500    0.90126477
## precisionMean    recallMean        f1Mean  accuracyMean       MCCMean
##    0.37075316    0.81040198    0.34659367    0.56662974    0.25617901
##       AUCMean       MSEMean
##    0.88914037    0.01333194
##
##  eve
##
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##
```

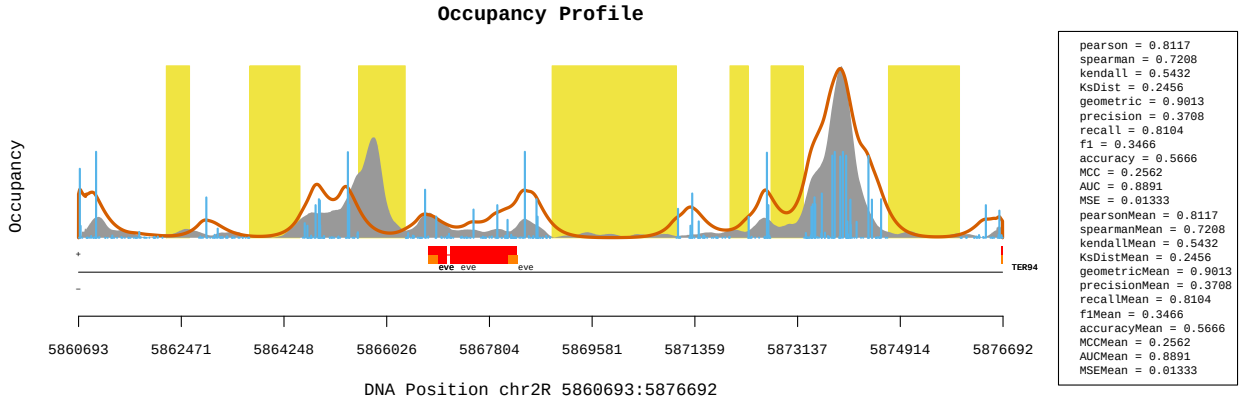
NOTE These functions can compute multiple parameter combinations if needed. `computeOptimal` is essentially a combination of these functions with a little more magic to it.

For more information on these functions see **Parameters Discription**

Step 7 - Plotting Single combination

Finally, you can plot your newly computed profiles.

```r
plotOccupancyProfile(predictedProfile=chip,
                     ChIPScore=eveLocusChip,
                     chromatinState=Access,
                     occupancy=occup,
                     goodnessOfFit=accu,
                     geneRef=geneRef)
```



In this case we have also added a gene reference object. This object is a GRanges object containing the postion of various *genetic elements* such as 3'UTR, 5'UTR, introns , etc

NOTE `plotOccupancyProfile` offers the possibility to customise graphical parameters. Unfortunately, `plotOptimalHeatMaps` offers limited graphical parameter customisation.

Parameter Description

In the following section, we will describe the different parameters present in both `parameterOptions` and `genomicProfiles`. Information concerning arguments to functions are described in the manual pages for each function.

As a reminder, here are the parameter options for the `parameterOptions` object. Parameters are divided into different categories depending on when they are required internally.

```
parameterOptions()
```

```
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##                         parameterOptions
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##  processingChIP options
##
##  chipMean:                                          200
##  chipSd:                                            200
##  chipSmooth:                                        250
##  lociWidth:                                       20000
##  noiseFilter:                                      zero
##
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##  processingChIP options Updated
##
##  maxSignal:                                           1
##  backgroundSignal:                                    0
##
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##  Position Weight Matrix Options
##
##  naturalLog:                                       TRUE
##  noOfSites:                                         all
```

```
##  PWMpseudocount:                                    1
##
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##  Genome Wide Score options
##
##  strandRule:                                      max
##  whichstrand:                                      +-
##  lambdaPWM:                                         1
##
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##  PWM Scores above Threshold options
##
##  strandRule:                                      max
##  whichstrand:                                      +-
##
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##  Occupancy options
##
##  Ploidy:                                            2
##  lambdaPWM:                                         1
##  boundMolecules:                                 1000
##  maxSignal:                                         1
##  backgroundSignal:                                  0
##
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
##  ChIP Profile options
##
##  chipMean:                                        200
##  chipSd:                                          200
##  chipSmooth:                                      250
##  stepSize:                                         10
##
## _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

- **chipMean**: Average ChIP peak width. Peak width is used during the smoothing of ChIP data.
- **chipSd**: Standard deviation of peak width. SD of peak width is used during the smoothing of ChIP data.
- **chipSmooth**: Window width used for ChIP data smoothing.
- **lociWidth**: When no loci are provided, ChIPanalyser will split ChIP data into bins of width equals to lociWidth.
- **noiseFilter**: Noise filter applied to ChIP data. ChIPanalyser provides four filters (zero, mean, median and sigmoid). Zero assigns 0 to any score below zero. Mean and median assign 0 to any score below the mean or median score. Sigmoid applies a weight to each ChIP score based on a logistic distribution. Scores above the 95th quantile will be weighted with a score between 1 and 2. Scores below the 95th quantile will be weighted with a score between 1 and 0.
- **maxSignal**: Maximum ChIP score after normalisation. Required in later step of the analysis. However, this score is computed internally and stored into the `ChIPscore` object , result of `processingChIP`.
- **backgroundSignal**: Average ChIP score after normalisation. Required in later step of the analysis. However, this score is computed internally and stored into the `ChIPscore` object , result of `processingChIP`.
- **naturalLog**: Log transform to be applied to PFM to PWM conversion. If `TRUE`, natural log will be used otherwise log2 will be used.
- **noOfSites**: Number of sites in the PWM that will be used for analysis. Default is set at “all” meaning all sites will be used. In the case that this argument is changed, ChIPanalyser requires a numeric value

describing the number of sites selected (from first site).

- **PWMpseudocount**: Pseudo-count used during PFM to PWM conversion.
- **strandRule**: PWM score computation mode. `max` returns the highest PWM score regardless of strand. `sum` returns the sum of PWM scores over both strands. `mean` returns the average PWM score oer both strands. It should be noted that this argument is only relevant when both strands are considered. See below.
- **whichstrand**: Strand that should be used for analysis. Options are: both `+-` or `-+`, plus only `+` or negative only `-`.
- **lambdaPWM**: Value to be assigned to λ . Positive numeric value.
- **ploidy**: Ploidy level of the organim used during analysis.
- **boundMolecules**: Number of molecules used to run analysis. Positive numeric value.

As you can see, some of these parameters are used during multiple steps of the analysis. If these paremeters have been changed in either a `parameterOptions` object or `genomicProfiles` object, please ensure that you parse these objects to each function. Each function will extract the values you have assigned to each parameter and use those values for analysis. It is possible to update, these parameters between each step of the analysis however, we recommend to set all parameters beforehand to avoid unwanted parameter mismatch.

Next, we will describe the content of `genomicProfiles` objects. As a reminder, `genomicProfiles` object have the following structure:

```
str(genomicProfiles())
```

```
## Formal class 'genomicProfiles' [package "ChIPanalyser"] with 31 slots
##   ..@ PWM               : num[0 , 0 ]
##   ..@ PFM               : num[0 , 0 ]
##   ..@ PFMFormat         : chr "raw"
##   ..@ BPFrequency       : num [1:4] 0.25 0.25 0.25 0.25
##   ..@ minPWMScore       : logi(0)
##   ..@ maxPWMScore       : logi(0)
##   ..@ profiles          :Formal class 'CompressedGRangesList' [package "GenomicRanges"] with 5 slots
##   .. .. ..@ unlistData     :Formal class 'GRanges' [package "GenomicRanges"] with 7 slots
##   .. .. .. .. ..@ seqnames       :Formal class 'Rle' [package "S4Vectors"] with 4 slots
##   .. .. .. .. .. .. ..@ values         : Factor w/ 0 levels:
##   .. .. .. .. .. .. ..@ lengths        : int(0)
##   .. .. .. .. .. .. ..@ elementMetadata: NULL
##   .. .. .. .. .. .. ..@ metadata       : list()
##   .. .. .. .. ..@ ranges         :Formal class 'IRanges' [package "IRanges"] with 6 slots
##   .. .. .. .. .. .. ..@ start          : int(0)
##   .. .. .. .. .. .. ..@ width          : int(0)
##   .. .. .. .. .. .. ..@ NAMES          : NULL
##   .. .. .. .. .. .. ..@ elementType    : chr "ANY"
##   .. .. .. .. .. .. ..@ elementMetadata: NULL
##   .. .. .. .. .. .. ..@ metadata       : list()
##   .. .. .. .. ..@ strand         :Formal class 'Rle' [package "S4Vectors"] with 4 slots
##   .. .. .. .. .. .. ..@ values         : Factor w/ 3 levels "+","-","*":
##   .. .. .. .. .. .. ..@ lengths        : int(0)
##   .. .. .. .. .. .. ..@ elementMetadata: NULL
##   .. .. .. .. .. .. ..@ metadata       : list()
##   .. .. .. .. ..@ seqinfo        :Formal class 'Seqinfo' [package "GenomeInfoDb"] with 4 slots
##   .. .. .. .. .. .. ..@ seqnames   : chr(0)
##   .. .. .. .. .. .. ..@ seqlengths : int(0)
##   .. .. .. .. .. .. ..@ is_circular: logi(0)
##   .. .. .. .. .. .. ..@ genome     : chr(0)
##   .. .. .. .. ..@ elementMetadata:Formal class 'DFrame' [package "S4Vectors"] with 6 slots
##   .. .. .. .. .. .. ..@ rownames       : NULL
```

```
##   .. .. .. .. .. .. ..@ nrows          : int 0
##   .. .. .. .. .. .. ..@ listData       : Named list()
##   .. .. .. .. .. .. ..@ elementType    : chr "ANY"
##   .. .. .. .. .. .. ..@ elementMetadata: NULL
##   .. .. .. .. .. .. ..@ metadata       : list()
##   .. .. .. .. ..@ elementType    : chr "ANY"
##   .. .. .. .. ..@ metadata       : list()
##   .. .. ..@ elementMetadata:Formal class 'DataFrame' [package "S4Vectors"] with 6 slots
##   .. .. .. .. ..@ rownames       : NULL
##   .. .. .. .. ..@ nrows          : int 0
##   .. .. .. .. ..@ listData       : Named list()
##   .. .. .. .. ..@ elementType    : chr "ANY"
##   .. .. .. .. ..@ elementMetadata: NULL
##   .. .. .. .. ..@ metadata       : list()
##   .. .. ..@ elementType    : chr "GRanges"
##   .. .. ..@ metadata       : list()
##   .. .. ..@ partitioning   :Formal class 'PartitioningByEnd' [package "IRanges"] with 5 slots
##   .. .. .. .. ..@ end            : int(0)
##   .. .. .. .. ..@ NAMES          : NULL
##   .. .. .. .. ..@ elementType    : chr "ANY"
##   .. .. .. .. ..@ elementMetadata: NULL
##   .. .. .. .. ..@ metadata       : list()
##   ..@ DNASequenceLength : logi(0)
##   ..@ averageExpPWMScore: logi(0)
##   ..@ ZeroBackground    : logi(0)
##   ..@ drop              : logi(0)
##   ..@ tags              : chr "empty"
##   ..@ ploidy            : num 2
##   ..@ boundMolecules    : num 1000
##   ..@ backgroundSignal  : num 0
##   ..@ maxSignal         : num 1
##   ..@ lociWidth         : num 20000
##   ..@ chipMean          : num 200
##   ..@ chipSd            : num 200
##   ..@ chipSmooth        : num 250
##   ..@ stepSize          : num 10
##   ..@ removeBackground  : num 0
##   ..@ noiseFilter       : chr "zero"
##   ..@ PWMThreshold      : num 0.7
##   ..@ strandRule        : chr "max"
##   ..@ whichstrand       : chr "+-"
##   ..@ lambdaPWM         : num 1
##   ..@ naturalLog        : logi TRUE
##   ..@ noOfSites         : chr "all"
##   ..@ PWMpseudocount    : num 1
##   ..@ paramTag          : chr "empty"
```

As you can see, we are using the structure function to show all internal slots. The `genomicProfiles` object inherit from `parameterOptions`, contain slots that are not user updatetable and finally the show method applied to `genomicProfiles` varies with each step of the analysis. This is intended to reduce information overload when "looking" at an object.

- **PWM**: A Position Weight Matrix. Either directly user provided or will be built internally if a Position Frequency Matrix is provided.
- **PFM**: A Position Frequency Matrix. This argument can either be a path towards a file containing the

PFM (RAW, JASPAR or Transfac format) or a matrix with rows being A, C, T, G and columns being PFM sites.

- **PFMFormat**: Format of provided PFM. `PFMFormat` can be one of the following: RAW, JASPAR, transfac or matrix. Matrix format is used if the provided PFM is an R matrix.
- **BPFrequency**: Genome alphabet Frequency. This parameter can be user provided in the form of a numeric vector (Frequency of each base pair in the following order A, C, T, G). For convenience, ChIPanalyser will automatically compute genome wide base pair frequency if a `DNAStringSet` object or `BSgenome` object is provided to this argument.
- **minPWMScore**: minimum PWM score over entire genome. Internally updated.
- **maxPWMScore**: maximum PWM score over entire genome. Internally updated.
- **profiles**: Storage slot for PWM scores above threshold, Occupancy, ChIP like profiles and Goodness of Fit. This slot holds the results each step of the analysis. Updated internally.
- **DNASequenceLength**: Length of DNA sequence used for analysis. Updated internally but can be provided by user.
- **averageExpPWMScore**: Average Exponetial PWM score. Updated Internally.
- **drop**: Regions that did not contain any accessible DNA or did not contain sites above threshold.

All other parameters have either been explained above or are part of the internal working of the package. These parameters are mainly used to keep track of the advancement of the analysis between each step. They should not be changed by user.

Finally, we provide a list of setter/getter functions for each slot:

```
## Accessors and Setters for parameterOptions and genomicProfiles
avrageExpPWMScore(obj)
backgroundSignal(obj)
backgroundSignal(obj)<-value
boundMolecules(obj)
boundMolecules(obj)<-value
BPFrequency(obj)
BPFrequency(obj)<-value
chipMean(obj)
chipMean(obj)<-value
chipSd(obj)
chipSd(obj)<-value
chipSmooth(obj)
chipSmooth(obj)<-value
DNASequenceLength(obj)
drop(obj)
lambdaPWM(obj)
lambdaPWM(obj)<-value
lociWidth(obj)
lociWidth(obj)<-value
maxPWMScore(obj)
maxSignal(obj)
maxSignal(obj)<-value
minPWMScore(obj)
naturalLog(obj)
naturalLog(obj)<-value
noiseFilter(obj)
noiseFilter(obj)<-value
noOfSites(obj)
noOfSites(obj)<-value
PFMFormat(obj)
PFMFormat(obj)<-value
```

```
    ploidy(obj)
    ploidy(obj)<-value
    PositionFrequencyMatrix(obj)
    PositionFrequencyMatrix(obj)<-value
    PositionWeightMatrix(obj)
    PositionWeightMatrix<-value
    profiles(obj)
    PWMpseudocount(obj)
    PWMpseudocount(obj)<-value
    PWMThreshold(obj)
    PWMThreshold(obj)<-value
    removeBackground(obj)
    removeBackground(obj)<-value

    stepSize(obj)
    stepSize(obj)<-value
    strandRule(obj)
    strandRule(obj)<-value
    whichstrand(obj)
    whichstrand(obj)<-value

    ## ChIPScore slots accessors
    loci(obj)
    scores(obj)
```

Session Info

```
sessionInfo()
```

```
## R version 3.6.1 (2019-07-05)
## 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_1.8.0
##  [3] RcppRoll_0.3.0
##  [4] BSgenome_1.54.0
##  [5] rtracklayer_1.46.0
```

```
##  [6] Biostrings_2.54.0
##  [7] XVector_0.26.0
##  [8] GenomicRanges_1.38.0
##  [9] GenomeInfoDb_1.22.0
## [10] IRanges_2.20.0
## [11] S4Vectors_0.24.0
## [12] BiocGenerics_0.32.0
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.2                  compiler_3.6.1
##  [3] BiocManager_1.30.9          bitops_1.0-6
##  [5] tools_3.6.1                 zlibbioc_1.32.0
##  [7] digest_0.6.22               evaluate_0.14
##  [9] lattice_0.20-38             rlang_0.4.1
## [11] Matrix_1.2-17               DelayedArray_0.12.0
## [13] yaml_2.2.0                  xfun_0.10
## [15] GenomeInfoDbData_1.2.2      stringr_1.4.0
## [17] knitr_1.25                  caTools_1.17.1.2
## [19] gtools_3.8.1                grid_3.6.1
## [21] Biobase_2.46.0              XML_3.98-1.20
## [23] BiocParallel_1.20.0         rmarkdown_1.16
## [25] gdata_2.18.0                ROCR_1.0-7
## [27] magrittr_1.5                gplots_3.0.1.1
## [29] Rsamtools_2.2.0             htmltools_0.4.0
## [31] matrixStats_0.55.0          GenomicAlignments_1.22.0
## [33] SummarizedExperiment_1.16.0 KernSmooth_2.23-16
## [35] stringi_1.4.3               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.