

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 thermodynamics as suggested by Zabet (Zabet et al. 2015). The statistical thermodynamics 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 :

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

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
if(!require("BSgenome.Dmelanogaster.UCSC.dm3", character.only = TRUE)){
  source("https://bioconductor.org/biocLite.R")
  biocLite("BSgenome.Dmelanogaster.UCSC.dm3")
}
library(BSgenome.Dmelanogaster.UCSC.dm3)

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

#Loading Position Frequency Matrix

PFM <- file.path(system.file("extdata",package="ChIPAnalyser"),"BCDS1x.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 DNASet 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 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 position 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
## Theta Threshold: 0.1

```

```

# 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
## Theta Threshold: 0.1
## Extracting ChIP score
eveLocusChip<-processingChIPseq(eveLocusChip,eveLocus,cores=1)
str(eveLocusChip)

## List of 2
## $ :List of 1
## ..$ eve: num [1:16000] 0.0116 0.0116 0.0116 0.0116 0.0116 ...
## $ :Formal class 'occupancyProfileParameters' [package "ChIPanalyser"] with 10 slots
## .. ..@ ploidy : num 2
## .. ..@ boundMolecules : num 1000
## .. ..@ backgroundSignal: num 0.118
## .. ..@ maxSignal : num 1.21
## .. ..@ chipMean : num 150
## .. ..@ chipSd : num 150
## .. ..@ chipSmooth : num 250
## .. ..@ stepSize : num 10
## .. ..@ removeBackground: num 0
## .. ..@ thetaThreshold : num 0.1

### 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 <- computeOptimal(DNASequenceSet = DNASequenceSet,
  genomicProfileParameters = GPP,
  LocusProfile = eveLocusChip,
  setSequence = eveLocus,
  DNAAccessibility = Access,
  occupancyProfileParameters = OPP,
  parameter = "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
## Extracting Optimal Set of Parameters

```

```
optimalParam
```

```

## $`Optimal Parameters`
## $`Optimal Parameters`$meanCorr
## [1] "1.25" "1000"
##
## $`Optimal Parameters`$meanMSE
## [1] "1"      "20000"
##
## $`Optimal Parameters`$meanTheta
## [1] "1"      "20000"
##
##
## $`Optimal Matrix`
## $`Optimal Matrix`$meanCorr
##           1          10          20          50          100          200          500
## 0.25 0.6602772 0.6576947 0.6551822 0.6514593 0.6494263 0.6499456 0.6555434
## 0.5   0.6535978 0.6536916 0.6539633 0.6543272 0.6554168 0.6574778 0.6617672
## 0.75 0.6583669 0.6613316 0.6633367 0.6676624 0.6723363 0.6882925 0.7242859
## 1     0.6918509 0.6961684 0.7001019 0.7063902 0.7129092 0.7060078 0.7214202
## 1.25 0.6110660 0.6146134 0.6505584 0.6576290 0.6659699 0.6776575 0.7269544
## 1.5   0.6703686 0.6719562 0.6819744 0.6866856 0.6934807 0.6947786 0.7149292
## 1.75 0.5398119 0.5414084 0.5401321 0.5485227 0.5574758 0.5624536 0.5881552
## 2     0.5383730 0.5394533 0.5405728 0.5344902 0.5376875 0.5518599 0.5660524
## 2.5   0.5334553 0.5339925 0.5345826 0.5385137 0.5382093 0.5454026 0.5768995
## 3     0.5526157 0.5529385 0.5532354 0.5541630 0.5555886 0.5583266 0.5649056
## 3.5   0.5526077 0.5528092 0.5530105 0.5536232 0.5545657 0.5563710 0.5612238
## 4     0.5526038 0.5527125 0.5528965 0.5532653 0.5539166 0.5551416 0.5585272
## 4.5   0.5525981 0.5526605 0.5527751 0.5530543 0.5535096 0.5543133 0.5567005
## 5     0.5525983 0.5526435 0.5527072 0.5529447 0.5532453 0.5538514 0.5554760
##           1000        2000        5000        10000        20000        50000        1e+05
## 0.25 0.6612732 0.6666572 0.6704267 0.6715311 0.6719736 0.6716491 0.6708537
## 0.5   0.6645271 0.6666766 0.6672102 0.6668480 0.7405238 0.6938537 0.7322566
## 0.75 0.7317366 0.7223656 0.7220206 0.7381301 0.7339830 0.7275176 0.7216003
## 1     0.7189104 0.7160194 0.7198667 0.7265780 0.7417883 0.7197661 0.7253244
## 1.25 0.7427727 0.7158401 0.7256048 0.7166612 0.6778048 0.6509658 0.6532002
## 1.5   0.6964649 0.6644996 0.6468762 0.6485881 0.6456311 0.6400614 0.6389072
## 1.75 0.6033746 0.6140456 0.6298019 0.6277120 0.6274524 0.6220929 0.6210012
## 2     0.5872667 0.6038863 0.6249853 0.6219062 0.6194208 0.6267141 0.6214853
## 2.5   0.5788600 0.5937492 0.6090590 0.6169536 0.6214966 0.6185399 0.6099407
## 3     0.5733246 0.5857457 0.6020406 0.6109613 0.6154832 0.6123745 0.6029286
## 3.5   0.5674280 0.5775429 0.5939363 0.6042836 0.6096137 0.6081928 0.5994755
## 4     0.5633210 0.5708024 0.5858795 0.5968793 0.6039616 0.6048698 0.5977263
## 4.5   0.5602584 0.5660522 0.5787111 0.5899323 0.5983345 0.6013779 0.5965013

```

```

## 5    0.5581661 0.5627272 0.5727846 0.5833755 0.5925049 0.5977771 0.5948087
##          2e+05      5e+05      1e+06
## 0.25 0.6701723 0.6681435 0.6662632
## 0.5  0.7328084 0.7203416 0.7211396
## 0.75 0.7100784 0.7326128 0.7199099
## 1    0.7012753 0.6612580 0.6530694
## 1.25 0.6457644 0.6358916 0.6349467
## 1.5  0.6358354 0.6184024 0.6154869
## 1.75 0.6217496 0.6088606 0.5938985
## 2    0.6121364 0.5940498 0.5782946
## 2.5  0.5962288 0.5774197 0.5671064
## 3    0.5897488 0.5736643 0.5646470
## 3.5  0.5875244 0.5725539 0.5648728
## 4    0.5871103 0.5726756 0.5647815
## 4.5  0.5875056 0.5733964 0.5650219
## 5    0.5873860 0.5738767 0.5656086
##
## $`Optimal Matrix`$meanMSE
##          1          10          20          50          100          200          500
## 0.25 31.35249 30.08925 28.83045 25.77580 22.32059 18.56625 14.80385
## 0.5  31.70697 30.78457 29.82872 27.33871 24.16417 20.10380 15.24265
## 0.75 31.29870 30.51660 29.69859 27.52328 24.63096 20.22500 14.11616
## 1    29.04385 28.27787 27.48690 25.42159 22.72385 18.94038 13.50113
## 1.25 29.90808 29.26872 28.33982 26.44963 23.83241 19.81041 13.32592
## 1.5  28.72232 28.16346 27.53043 25.89930 23.44603 20.04732 14.34872
## 1.75 28.56064 28.07982 27.55512 26.29844 24.13915 21.47974 17.11690
## 2    28.72967 28.38701 28.01640 26.97688 25.40600 23.01586 20.27060
## 2.5  29.03263 28.85912 28.67042 28.11267 27.30766 25.94672 23.05035
## 3    28.54996 28.44567 28.33132 27.99781 27.47291 26.53341 24.49329
## 3.5  28.55466 28.49220 28.42332 28.22007 27.89234 27.27700 25.72953
## 4    28.55729 28.51834 28.47526 28.34732 28.13833 27.73601 26.64890
## 4.5  28.55881 28.53340 28.50527 28.42139 28.28340 28.01407 27.25787
## 5    28.55971 28.54240 28.52321 28.46588 28.37117 28.18483 27.65007
##          1000          2000          5000          10000          20000          50000          1e+05
## 0.25 13.484804 13.120007 13.386717 13.830458 14.213159 14.247939 13.768571
## 0.5  13.244282 12.502590 11.996862 11.386446 10.565609 9.884869 9.501761
## 0.75 10.826872 9.699379 9.223615 9.174281 8.868910 8.775578 8.803997
## 1    10.634559 9.295550 8.771268 8.183135 7.881857 8.652990 9.940928
## 1.25 9.996224 8.468282 8.463150 9.015329 10.657890 14.486401 19.349303
## 1.5  11.481143 10.497415 11.428782 13.683235 18.881067 26.692340 33.745692
## 1.75 14.506248 15.013995 18.628136 22.734704 27.524053 37.182749 50.071231
## 2    19.177055 19.905509 21.956649 29.766182 40.674582 51.897423 58.932794
## 2.5  23.597109 26.629879 33.315321 39.093762 45.846621 58.062193 70.930161
## 3    23.128033 23.777787 28.752372 35.393122 44.360480 61.219281 79.132119
## 3.5  24.013087 22.986631 25.201721 30.814037 40.346231 60.090664 81.450402
## 4    25.203460 23.471289 23.352458 26.861401 35.177603 55.514297 78.804622
## 4.5  26.160901 24.516694 22.898358 24.340872 30.392962 49.132399 72.878903
## 5    26.836749 25.482183 23.241248 23.182488 26.844285 42.497391 65.292096
##          2e+05          5e+05          1e+06
## 0.25 12.878622 11.476124 10.662333
## 0.5  9.040849 8.917573 8.980269
## 0.75 8.612272 8.617836 9.657544
## 1    12.719551 18.379186 23.542837
## 1.25 27.211320 35.706934 40.571820

```

```

## 1.5 38.754687 52.997086 63.881060
## 1.75 59.240780 71.924429 88.102775
## 2 68.730385 90.013606 114.014844
## 2.5 89.151113 123.649857 152.571045
## 3 103.241001 143.933094 160.593223
## 3.5 109.169661 152.772834 159.388620
## 4 108.815048 154.497605 158.276174
## 4.5 104.263303 151.902516 156.519702
## 5 97.139645 146.590593 153.939052
##
## $`Optimal Matrix`$meanTheta
## 1 10 20 50 100 200
## 0.25 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 0.5 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 0.75 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 1 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 1.25 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 1.5 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 1.75 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 2 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 2.5 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 3 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 3.5 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 4 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 4.5 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 5 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 500 1000 2000 5000 10000 20000
## 0.25 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 0.5 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903 0.07008813
## 0.75 0.06797851 0.06867779 0.07447545 0.07827957 0.08045645 0.08275911
## 1 0.06774903 0.06787691 0.07765447 0.08229608 0.08878969 0.09411340
## 1.25 0.06822896 0.07430533 0.08524054 0.08573696 0.08006818 0.06774903
## 1.5 0.06774903 0.06774903 0.06876369 0.06774903 0.06774903 0.06774903
## 1.75 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 2 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 2.5 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 3 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 3.5 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 4 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 4.5 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 5 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 50000 1e+05 2e+05 5e+05 1e+06
## 0.25 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 0.5 0.07302484 0.07706535 0.08105527 0.08094590 0.08038077
## 0.75 0.08290252 0.08199014 0.08381539 0.08501123 0.07474374
## 1 0.08342099 0.07296345 0.06774903 0.06774903 0.06774903
## 1.25 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 1.5 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 1.75 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 2 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 2.5 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 3 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 3.5 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 4 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903

```



```
## 4.5 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
## 5 0.06774903 0.06774903 0.06774903 0.06774903 0.06774903
##
##
## $Parameter
## [1] "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 DNAAccessibility
##      <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 DNAAccessibility
##      <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 0.0977045894468306
## eve 0.0976612507347727
## eve 0.154456350403535
## eve 0.100529067610285
## eve 0.0976601353676412
## ...      ...
## eve 0.105494267881878
## eve 0.0979488725093411
## eve 0.0976608621055203
## eve 0.0982475912592569
## eve 0.0976895497330001
##
## -----
## 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    * | 0.0644413047696366
## eve   chr2R  5860703-5860713    * | 0.0698520515099746
## eve   chr2R  5860713-5860723    * | 0.0754053510276315
## eve   chr2R  5860723-5860733    * | 0.0811258937963851
## eve   chr2R  5860733-5860743    * | 0.0870391138686896
## ...     ...                ...   ... .          ...
## eve   chr2R  5876643-5876653    * | 0.083597878173792
## eve   chr2R  5876653-5876663    * | 0.0796242551641794
## eve   chr2R  5876663-5876673    * | 0.0755554966819536
## eve   chr2R  5876673-5876683    * | 0.0713735126575455
## eve   chr2R  5876683-5876693    * | 0.0670597096103467
## -----
## 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 $n = \text{stepSize}$, see below).

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

In order to plot the model accuracy (predicted model against real ChIP-seq data).

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

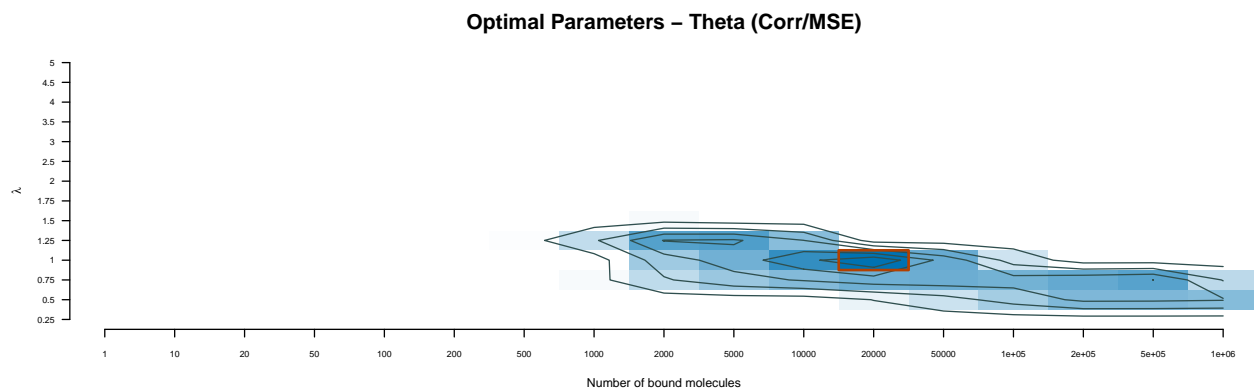
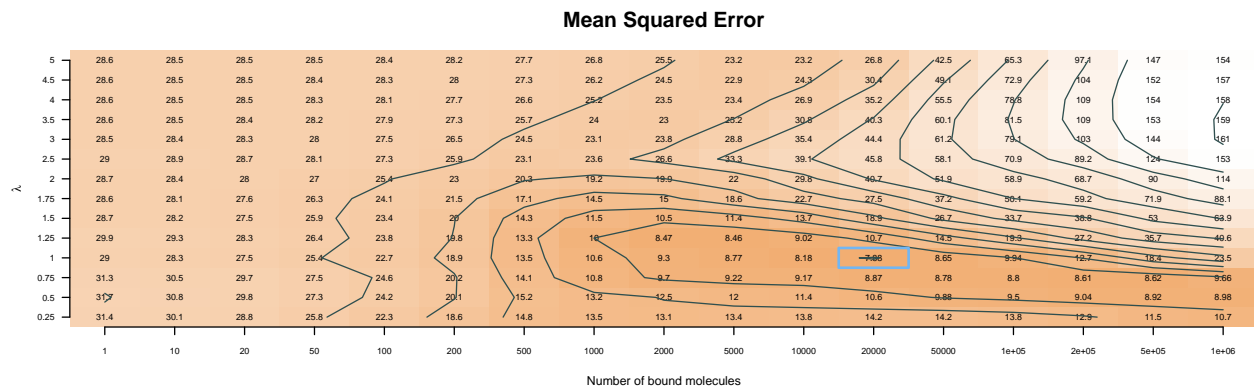
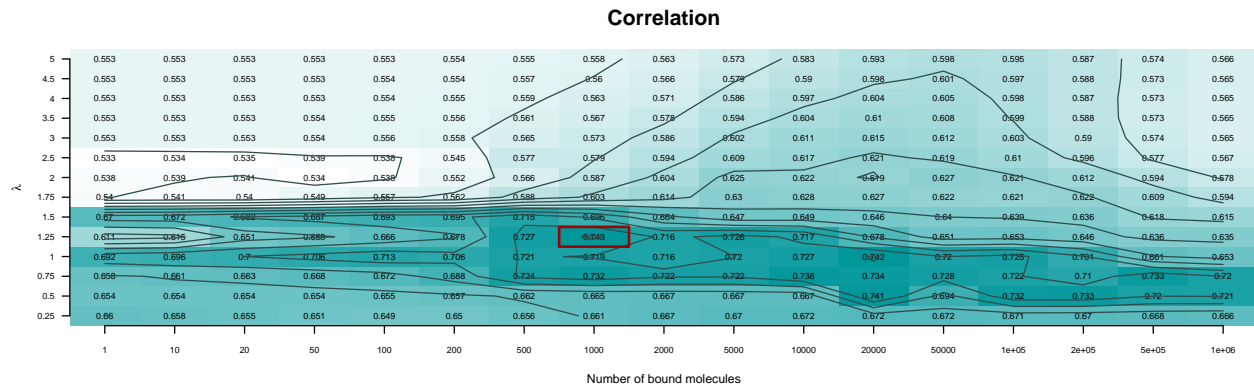
```
## $`lambda = 1.5 & boundMolecules = 1000`
## $`lambda = 1.5 & boundMolecules = 1000`$eve
```

```
##          Corr          MSE      meanCorr      meanMSE      meanTheta
## 0.69646492 0.01148114 0.69646492 11.48114288 0.06066164
```

Step 7 - Plotting

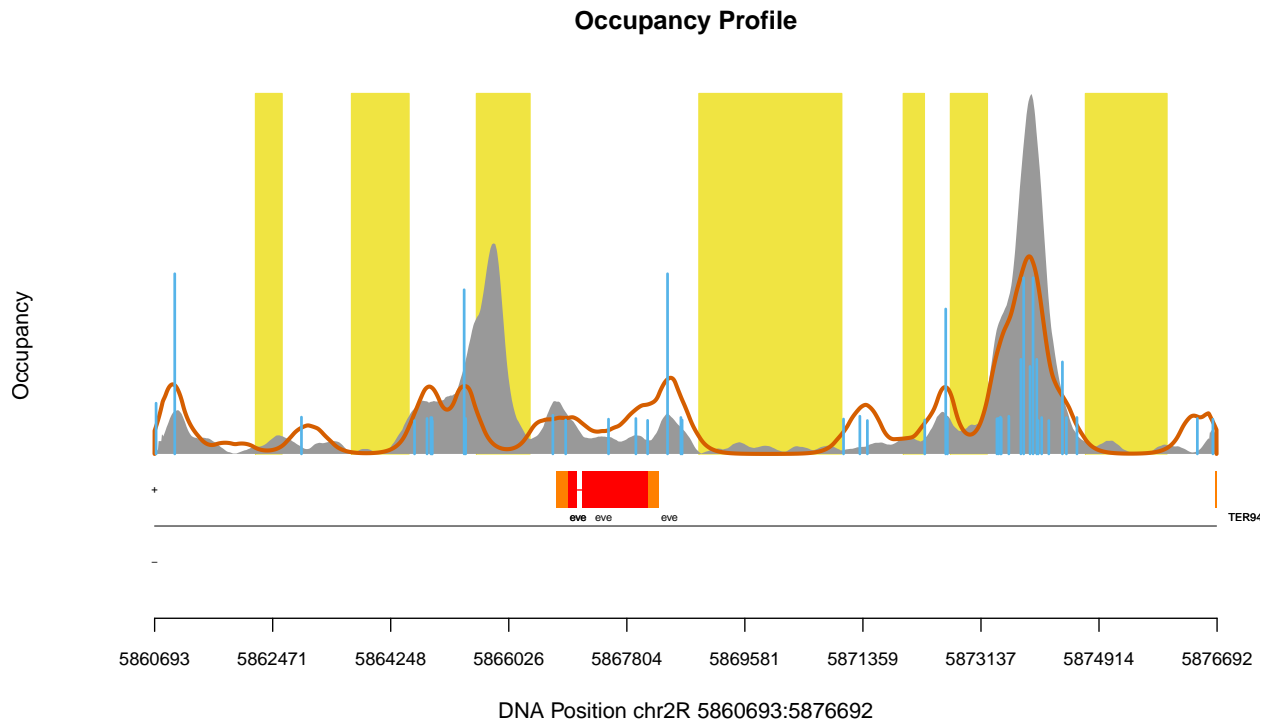
Finally, once all has been computed, it is possible to plot the results.

```
# Plotting Optimal heat maps
plotOptimalHeatMaps(optimalParam, parameter="all")
```



```
# Plotting occupancy Profile
##
plotOccupancyProfile(predictedProfile=chipProfile[[1]][[1]],
  setSequence=eveLocus,
  chipProfile = eveLocusChip[[1]],
```

```
DNAAccessibility = Access,
occupancy = AllSitesAboveThreshold(Occupancy)[[1]][[1]],
occupancyProfileParameters = OPP,
geneRef = geneRef)
```



```
## [1] 1
## exon[1] 2
## five_prime_UTR[1] 3
## intron[1] 4
## three_prime_UTR
```

Work Flow - Full Guide

This section will describe ChIPanalyzer'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 ChIPanalyzer 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 below 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 Matrices 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 below.

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 **DNAStrngSet**. 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 PWM Scores 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 PWM Scores 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 be 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 not 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 stored 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` includes 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 DNAAccessibility  
##      <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 0.0977045894468306
## eve 0.0976612507347727
## eve 0.154456350403535
## eve 0.100529067610285
## eve 0.0976601353676412
## ...
## eve 0.105494267881878
## eve 0.0979488725093411
## eve 0.0976608621055203
## eve 0.0982475912592569
## eve 0.0976895497330001
##
## -----
## 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 DNAAccessibility
##      <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 0.0977045894468306
## eve 0.0976612507347727
## eve 0.154456350403535
## eve 0.100529067610285
## eve 0.0976601353676412
## ...
## eve 0.105494267881878
## eve 0.0979488725093411
## eve 0.0976608621055203
## eve 0.0982475912592569
## eve 0.0976895497330001
##
## -----
## seqinfo: 1 sequence from an unspecified genome; no seqlengths

```

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 (**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 your 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 `processingChIPseq` below ****

```
OPP <- occupancyProfileParameters(ploidy = 2 ,boundMolecules = 1000 ,
  backgroundSignal = 0 ,maxSignal = 1, chipMean = 150 , chipSd = 150 ,
  chipSmooth = 250 , stepSize = 10 ,
  removeBackground = 0 , thetaThreshold = 0.1)
```

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 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 signal 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
```

- `thetaThreshold`, a numeric value describing the threshold used to calculate our in house *theta* value (Default is set at 0.1). *Theta* is a metric used to demonstrate which parameters are optimal by maximising the correlation and minimising the Mean Squared Error (MSE) between the predicted profile and actual ChIP-seq profiles. The higher the value of *theta*, the better the ratio between correlation and MSE. Values below this threshold are discarded (replaced by Threshold) as they represent extremely poor accuracy with actual ChIP-seq data.

```
thetaThreshold(OPP)
```

```
thetaThreshold(OPP) <- 0.1
```


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 provide 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 built or updated (see above) with value for the relevant slots (See `backgroundSignal` and `maxSignal`).

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

As input, this function 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 signal 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 aforementioned object. `processingChIPseq` will return an 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 accessible DNA for a given organism/cell type. If `reduce` is `NULL`, only regions with accessible DNA will be selected. If `peaks` is not `NULL`, regions with peaks *and* accessible DNA will be selected.
- `cores` is the number of cores 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 function 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 DNASet instance of length 15
##      width seq                      names
## [1] 23011544 CGACAATGCACGACAGAGG...ATGAACCCCTTTCAAA chr2L
## [2] 21146708 GACCCGCTAGGAGATGTTG...TTGCATTCTAGGAATTC chr2R
## [3] 24543557 TAGGGAGAAATATGATCGC...AACCAAGTTAATGTTCCG chr3L
## [4] 27905053 GAATTCTCTCTGTTGTAG...TTCGCATTCTAGGAATTC chr3R
## [5] 1351857 GAATTCGCGTCCGCTTACC...CGATTGAGATATATGAA chr4
## ...
## [11] 2555491 AACGAGGCCCATTTTCATAC...ATGCCATTTCGCTAGAAGT chr3LHet
## [12] 2517507 CCCTGTTTGCATCAGCGTT...TAAAAACAATTTGCTCCC chr3RHet
## [13] 204112 TAGATAGATAGATAGATAG...ATCGAGTTAATGTTTGC chrXHet
## [14] 347038 AGGTCACGTAATGCTGAT...TTGTTTCCCGGGATTG chrYHet
## [15] 29004656 ATTGAAAATGGATTGCATT...CAAGACCTTTCAAGACAA chrUextra
```

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

```
#Extracting DNASet 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 be 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 `ChIPAnalyser`. 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 object from one functions 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 `computeOccupancy` 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 fit. 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  
##  
## Floidy: 2  
## boundMolecules: 1000  
## backgroundSignal: 0.117869636963696  
## maxSignal: 1.207  
## chipMean: 150  
## chipSd: 150  
## chipSmooth: 250  
## Step Size: 10  
## Theta Threshold: 0.1
```

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 DNAAccessibility
##      <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 0.0977045894468306
## eve 0.0976612507347727
## eve 0.154456350403535
## eve 0.100529067610285
## eve 0.0976601353676412
## ...      ...
## eve 0.105494267881878
## eve 0.0979488725093411
## eve 0.0976608621055203
## eve 0.0982475912592569
## eve 0.0976895497330001
##
## -----
## 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 * | 0.0644413047696366
## eve chr2R 5860703-5860713 * | 0.0698520515099746
## eve chr2R 5860713-5860723 * | 0.0754053510276315
## eve chr2R 5860723-5860733 * | 0.0811258937963851
## eve chr2R 5860733-5860743 * | 0.0870391138686896
## ...      ...      ...      ... .      ...
## eve chr2R 5876643-5876653 * | 0.083597878173792
## eve chr2R 5876653-5876663 * | 0.0796242551641794
## eve chr2R 5876663-5876673 * | 0.0755554966819536
## eve chr2R 5876673-5876683 * | 0.0713735126575455
## eve chr2R 5876683-5876693 * | 0.0670597096103467
## -----
## 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)
```

Input data - Accuracy Estimate

`profileAccuracyEstimate` requires only three arguments. `precitedProfile` is the result of `computeChipProfile` and `occupancyProfileParameters` is a `occupancyProfileParameters`. Finally, `LocusProfile` is a list containing actual ChIP-seq profiles. These profiles should be normalised to a base pair level. In other words, a peak should be divided by its width. 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] 0.0116 0.0116 0.0116 0.0116 0.0116 ...
```

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.

profileAccuracyEstimate

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

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

The result of this function will be a list of accuracy estimates for every loci and every combination of `ScalingFactorPWM` and `boundMolecules`. The correlation and Mean Squared Error (MSE) represents the correlation and MSE between the predicted profile (for a given combination on `lambda` and `boundMolecules`)

and the ChIP-seq profile for the same loci. `meanCorr` and `meanMSE` describe the average correlation and MSE for all loci (for a given combination on `ScalingFactorPWM` and `boundMolecules`). The idea behind average correlation and MSE is that the scaling factor and number of molecules should be the same regardless of the loci as all TF's are contained within the same nucleus. Finally, `meanTheta` is an in house metric describing a modified ratio of correlation over MSE. The goal is to find the sweet spot between high correlation and low MSE (see `computeOptimal` and `plotOptimalHeatMaps`).

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`, this function will return the combination with the highest correlation, lowest Mean Squared Error or highest theta depending on which parameter was selected. As a reminder, theta is an in house metric representing a modified ratio of correlation over MSE (extreme values are replaced by threshold). The goal is to find the sweet spot between high correlation and low MSE. 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,
  parameter = "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 is extremely similar to the functions described above. As a quick reminder:

- `DNASequenceSet`, a `DNAStringSet` (or `BSSgenome`) 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`

`parameter` defines which metric you wish to compute. There are four possible choices: *correlation*, *MSE*, *theta* 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,  
  parameter = "all",  
  cores=1)  
optimalParam
```

This functions returns either a list or a list of lists (if “all” parameter was selected). Each element in the list represents the **optimal set of parameters**, the **optimal matrix** (a matrix with correlation, MSE and/or theta computed for a given combination of `ScalingFactorPWM` and `boundMolecules`) and finally the **selected parameter**.

Plotting Results

As it is the case in many 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 ,  
  parameter="all", Contour=TRUE)
```

Input Data & Plotting

`plotOptimalHeatMaps` only requires one data input in the form of the result of `computeOptimal` (see `computeOptimal`). The `parameter` argument defines which of the following parameters you wish to plot: *correlation*, *MSE*, *theta* or *all of the previous*. Finally, `Contour` defines if you which to plot Contour lines on your heat map. As an example:

```
plotOptimalHeatMaps(optimalParam, parameter="all")
```

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,
  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 3.5.0 (2018-04-23)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 16.04.4 LTS
##
## Matrix products: default
## BLAS: /home/biocbuild/bbs-3.7-bioc/R/lib/libRblas.so
## LAPACK: /home/biocbuild/bbs-3.7-bioc/R/lib/libRlapack.so
##
## locale:
## [1] LC_CTYPE=en_US.UTF-8      LC_NUMERIC=C
## [3] LC_TIME=en_US.UTF-8      LC_COLLATE=C
## [5] LC_MONETARY=en_US.UTF-8  LC_MESSAGES=en_US.UTF-8
## [7] LC_PAPER=en_US.UTF-8    LC_NAME=C
## [9] LC_ADDRESS=C            LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## 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_1.2.0
```

```

## [3] RcppRoll_0.2.2
## [4] BSgenome_1.48.0
## [5] rtracklayer_1.40.0
## [6] Biostrings_2.48.0
## [7] XVector_0.20.0
## [8] GenomicRanges_1.32.0
## [9] GenomeInfoDb_1.16.0
## [10] IRanges_2.14.0
## [11] S4Vectors_0.18.0
## [12] BiocGenerics_0.26.0
##
## loaded via a namespace (and not attached):
## [1] Rcpp_0.12.16          knitr_1.20
## [3] magrittr_1.5          GenomicAlignments_1.16.0
## [5] zlibbioc_1.26.0      BiocParallel_1.14.0
## [7] lattice_0.20-35      stringr_1.3.0
## [9] tools_3.5.0          grid_3.5.0
## [11] SummarizedExperiment_1.10.0 Biobase_2.40.0
## [13] matrixStats_0.53.1   htmltools_0.3.6
## [15] yaml_2.1.18          rprojroot_1.3-2
## [17] digest_0.6.15        Matrix_1.2-14
## [19] GenomeInfoDbData_1.1.0 bitops_1.0-6
## [21] RCurl_1.95-4.10     evaluate_0.10.1
## [23] rmarkdown_1.9        DelayedArray_0.6.0
## [25] stringi_1.1.7        compiler_3.5.0
## [27] Rsamtools_1.32.0     backports_1.1.2
## [29] XML_3.98-1.11

```

References

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