The ChIPanalyser User's Guide

Patrick Martin

14/08/2017

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 fashion. 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 generatesa genome wide profile with exhibiting 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 predicting Transcription factor binding. At the core of this packageresides an approximation of statistical thermo dynamcis as suggested by Zabet (Zabet et al. 2015). The statistical thermo dynamcis framework proposed by Zabet offers a strong ground for binding site prediction as it requires minimal data input. In its current version, ChIPAnalyser requires a DNA sequence, a Position Weight Matrix, the number of bound molecules (or TFs bound to DNA) and a scaling factor for TF specificity. To improve the accuracy of the model, it is also possible to incorporate DNA accessibility data.

Methods

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

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

with

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

Work Flow - Quick start

Example data Loading

Before going through the inner workings of the package and the work flow, this section will quickly demonstrate how to load example datasets stored in the package. This data represents 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`).

```r
library(ChIPanalyser)
```

```
## Loading required package: GenomicRanges

## Loading required package: stats4

## Loading required package: BiocGenerics

## Loading required package: parallel

##
## Attaching package: 'BiocGenerics'

## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB

## The following objects are masked from 'package:stats':
##
##     IQR, mad, xtabs

## The following objects are masked from 'package:base':
##
##     anyDuplicated, append, as.data.frame, cbind, colnames,
##     do.call, duplicated, eval, evalq, Filter, Find, get, grep,
##     grepl, intersect, is.unsorted, lapply, lengths, Map, mapply,
##     match, mget, order, paste, pmax, pmax.int, pmin, pmin.int,
##     Position, rank, rbind, Reduce, rownames, sapply, setdiff,
##     sort, table, tapply, union, unique, unsplit, which, which.max,
##     which.min

## Loading required package: S4Vectors

##
## Attaching package: 'S4Vectors'

## The following objects are masked from 'package:base':
##
##     colMeans, colSums, expand.grid, rowMeans, rowSums

## Loading required package: IRanges

## Loading required package: GenomeInfoDb

## Loading required package: Biostrings

## Loading required package: XVector

## Loading required package: BSgenome

## Loading required package: rtracklayer
```

```r
#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")
    }
```

```
## Loading required package: 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"),"BCDSlx.pfm")

#Checking if correctly loaded
ls()
```

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

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

- `DNASequenceSet` is DNAStringSet extracted from the *Drosophila melanogaster* genome (BSgenome). It is advised to use a full genome sequence for this object.
- `PFM` is a path to file. In this case, it is a Position Frequency Matrix derived from the Bicoid Transcription factor in *Drosophila melanogaster*. This PFM is in RAW format.

- `Access` is a `GRanges` object containing accessible DNA for the sequence above.
- `geneRef` is list of `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 list containing real ChIP-seq data (normalised to each base pair) of the eve strip locus in *Drosophila melanogaster*.

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

Quick Start

Step 1 - Building Data objects

The first step is to set up your data storing objects. These objects will automatically compute Position Weight Matrix from a Position Frequency Matrix, and Base Pair Frequency from a DNAStringSet. The values that are provided in this example are extracted from the biological data used in this vignette. These values will differ depending on the source of the data and the data itself.

```
# Building a genomicProfileParameters objects for data
# storage and PWM computation
GPP <- genomicProfileParameters(PFM=PFM,
                                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
```

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

```
## 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
```

Step 2 - Optimal Parameters

The model is based on the approximation of statistical thermodynamics with inference of two parameters (ScalingFactorPWM and boundMolecules). `ScalingFactorPWM` (described in the model as lambda). In order to backward 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 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)
```

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

```
## Computing Genome Wide PWM Score

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

## Computing Occupancy

## Computing ChIP-seq-like Profile

## Computing Accuracy of Profile

## Extracting Optimal Set of Parameters
```

```
optimalParam
```

```
## $`Optimal Parameters`
## $`Optimal Parameters`$meanCorr
## [1] "1"     "20000"
##
## $`Optimal Parameters`$meanMSE
## [1] "1.5"  "1000"
##
## $`Optimal Parameters`$meanTheta
## [1] "1.5"  "1000"
##
##
## $`Optimal Matrix`
## $`Optimal Matrix`$meanCorr
##              1        10        20        50       100       200       500
## 0.25 0.8173071 0.7648625 0.7324056 0.6998384 0.6918641 0.6980422 0.7225866
## 0.5  0.8124621 0.7939262 0.7782588 0.7571666 0.7480763 0.7470883 0.7551821
## 0.75 0.8183404 0.8161220 0.8077229 0.7912216 0.7814856 0.7806790 0.7864082
## 1    0.7233945 0.7600240 0.7795050 0.7964159 0.7989738 0.8032896 0.8061604
## 1.25 0.7384078 0.7663945 0.8026240 0.8198551 0.8232073 0.8227882 0.8262566
## 1.5  0.7323251 0.7522930 0.7691601 0.7967564 0.8150243 0.8223581 0.8270312
## 1.75 0.6024921 0.6254769 0.6431430 0.6928116 0.7391694 0.7785201 0.8029834
## 2    0.4631645 0.4836447 0.5045835 0.5549825 0.6154599 0.6833449 0.7519283
## 2.5  0.3334889 0.3428032 0.3528677 0.3794987 0.4193459 0.4871190 0.6188883
## 3    0.3462024 0.3517040 0.3577060 0.3750320 0.4017676 0.4481136 0.5452402
## 3.5  0.3459809 0.3495189 0.3534002 0.3647369 0.3826490 0.4150887 0.4906003
## 4    0.3458453 0.3481755 0.3507408 0.3582891 0.3703925 0.3929218 0.4490845
## 4.5  0.3457618 0.3473455 0.3490929 0.3542592 0.3626224 0.3784672 0.4197817
## 5    0.3457092 0.3468219 0.3480517 0.3516989 0.3576409 0.3690324 0.3996483
##           1000      2000      5000     10000     20000     50000     1e+05
## 0.25 0.7391873 0.7509182 0.7555294 0.7528503 0.7497344 0.7524092 0.7628162
## 0.5  0.7643083 0.7743409 0.7918896 0.8084623 0.8233247 0.8339117 0.8364327
## 0.75 0.7969066 0.8087598 0.8240734 0.8309887 0.8350043 0.8355943 0.8341099
## 1    0.8143036 0.8224855 0.8316129 0.8365528 0.8385134 0.8358283 0.8315642
## 1.25 0.8290259 0.8322070 0.8359957 0.8376988 0.8344907 0.8234591 0.8076988
## 1.5  0.8283286 0.8298271 0.8328232 0.8309558 0.8220926 0.8018848 0.7805414
## 1.75 0.8163186 0.8193007 0.8221161 0.8178857 0.8077554 0.7801782 0.7530692
## 2    0.7863337 0.8035143 0.8130667 0.8061440 0.7889372 0.7569877 0.7271277
## 2.5  0.6892226 0.7425114 0.7745917 0.7761068 0.7611964 0.7203457 0.6763833
## 3    0.6305303 0.6991009 0.7455816 0.7513856 0.7363326 0.6892882 0.6373771
## 3.5  0.5691911 0.6453690 0.7085259 0.7228507 0.7122580 0.6659814 0.6123856
## 4    0.5150263 0.5898400 0.6658995 0.6907780 0.6879340 0.6477956 0.5965755
## 4.5  0.4724551 0.5398032 0.6216342 0.6563457 0.6626286 0.6317972 0.5854277
## 5    0.4409555 0.4985991 0.5795785 0.6214853 0.6365423 0.6162424 0.5761141
##          2e+05     5e+05     1e+06
## 0.25 0.7805763 0.8081670 0.8239444
## 0.5  0.8375767 0.8359647 0.8332134
## 0.75 0.8332313 0.8323116 0.8268102
## 1    0.8208644 0.7978678 0.7795452
## 1.25 0.7867841 0.7631412 0.7491300
## 1.5  0.7603043 0.7283938 0.6999919
## 1.75 0.7253332 0.6791652 0.6301245
## 2    0.6900334 0.6238349 0.5625819
## 2.5  0.6213661 0.5369745 0.4759032
## 3    0.5754351 0.4909667 0.4376601
## 3.5  0.5497811 0.4690785 0.4212562
## 4    0.5360352 0.4590218 0.4142797
## 4.5  0.5283820 0.4547060 0.4116587
## 5    0.5234964 0.4531437 0.4111479
##
## $`Optimal Matrix`$meanMSE
##             1       10       20       50        100        200        500
## 0.25 15.04685 13.97562 12.95156 10.63547  8.385438  6.697997  6.900416
## 0.5  15.11895 14.32120 13.50855 11.48094  9.128463  6.667597  5.432525
## 0.75 15.09627 14.42898 13.74289 11.97172  9.757843  7.043254  4.707458
## 1    14.86914 14.22193 13.59179 12.01213 10.007746  7.380250  4.535701
## 1.25 14.96503 14.42754 13.84213 12.44074 10.617348  8.078654  4.675202
## 1.5  14.85616 14.38394 13.89923 12.68027 11.065254  8.739867  5.161890
## 1.75 14.80645 14.40054 13.99968 12.95127 11.557182  9.477683  5.977594
## 2    14.76671 14.41727 14.04926 13.11264 11.904359 10.109693  6.928287
## 2.5  14.81041 14.61253 14.39445 13.74620 12.728283 11.347832  8.798026
## 3    14.75344 14.63158 14.49688 14.09747 13.449090 12.452768 10.420109
## 3.5  14.75892 14.68619 14.60556 14.36501 13.968989 13.214925 11.689017
## 4    14.76199 14.71674 14.66652 14.51630 14.267440 13.776593 12.596019
## 4.5  14.76374 14.73429 14.70158 14.60362 14.440872 14.117811 13.210269
## 5    14.76479 14.74474 14.72248 14.65575 14.544754 14.323732 13.671296
##           1000      2000      5000     10000     20000     50000     1e+05
## 0.25  8.776980 11.054976 13.806813 15.509365 16.672284 16.973742 16.137618
## 0.5   6.782277  8.941916 10.643901 10.585541 10.011142  9.451026  9.343135
## 0.75  5.113025  6.629664  8.185566  8.787006  9.091635  9.469223  9.855875
## 1     4.248648  5.444165  7.361291  8.422404  9.289120 10.808115 12.771186
## 1.25  3.754813  4.629331  7.013696  8.764623 10.777319 14.847776 19.801733
## 1.5   3.723466  4.270712  7.025620  9.823153 13.636180 20.792376 27.871749
## 1.75  4.081700  4.238196  7.499015 11.407707 16.612189 26.661320 36.581710
## 2     4.806457  4.394454  7.455223 12.672217 20.280731 33.271254 44.932956
## 2.5   6.718845  5.566429  8.005080 13.852390 23.326730 41.709706 60.442992
## 3     8.339585  6.375130  6.714039 11.858543 22.590502 45.935978 70.721884
## 3.5   9.914196  7.782340  6.308824  9.429959 19.294588 44.902208 73.639455
## 4    11.185455  9.244621  6.824691  7.758454 15.135159 39.893477 70.475129
## 4.5  12.124242 10.489805  7.829615  7.185256 11.618750 33.012250 63.404005
## 5    12.789537 11.459082  8.933336  7.425353  9.361984 26.091998 54.474155
##          2e+05      5e+05      1e+06
## 0.25  14.535854  12.089027  10.705146
## 0.5    9.320336   9.421976   9.621293
## 0.75  10.412746  11.845658  14.055565
## 1     16.165841  22.808120  28.222222
## 1.25  26.201405  33.976871  39.071702
## 1.5   35.022343  46.457502  57.793239
## 1.75  47.152209  65.071222  85.289747
## 2     59.479282  87.087708 116.112721
## 2.5   84.862802 127.682240 165.038788
## 3    102.222972 152.285877 190.165648
## 3.5  109.550112 163.124991 192.639851
## 4    109.157106 165.361264 190.712517
## 4.5  103.615857 162.347568 188.178087
## 5     94.949130 156.013248 184.680569
##
## $`Optimal Matrix`$meanTheta
##              1        10        20        50       100       200       500
## 0.25 0.1479306 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281
## 0.5  0.1470537 0.1436987 0.1426281 0.1426281 0.1426281 0.1426281 0.1450543
## 0.75 0.1481176 0.1477161 0.1461959 0.1432092 0.1426281 0.1426281 0.1673963
## 1    0.1426281 0.1426281 0.1426281 0.1441493 0.1446123 0.1453935 0.1777367
## 1.25 0.1426281 0.1426281 0.1452730 0.1483918 0.1489985 0.1489227 0.1767317
## 1.5  0.1426281 0.1426281 0.1426281 0.1442110 0.1475174 0.1488448 0.1602187
## 1.75 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1453381
## 2    0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281
## 2.5  0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281
## 3    0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281
## 3.5  0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281
## 4    0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281
## 4.5  0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281
## 5    0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281
##           1000      2000      5000     10000     20000     50000     1e+05
## 0.25 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281
## 0.5  0.1426281 0.1426281 0.1433301 0.1463297 0.1490198 0.1509360 0.1513923
## 0.75 0.1558581 0.1463836 0.1491553 0.1504069 0.1511338 0.1512406 0.1509719
## 1    0.1916618 0.1510765 0.1505199 0.1514140 0.1517689 0.1512829 0.1505111
## 1.25 0.2207902 0.1797683 0.1513132 0.1516215 0.1510408 0.1490441 0.1461915
## 1.5  0.2224617 0.1943065 0.1507390 0.1504010 0.1487968 0.1451392 0.1426281
## 1.75 0.1999948 0.1933135 0.1488010 0.1480353 0.1462018 0.1426281 0.1426281
## 2    0.1639484 0.1828474 0.1471631 0.1459101 0.1427957 0.1426281 0.1426281
## 2.5  0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281
## 3    0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281
## 3.5  0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281
## 4    0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281
## 4.5  0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281
## 5    0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281 0.1426281
##          2e+05     5e+05     1e+06
## 0.25 0.1426281 0.1462763 0.1491320
## 0.5  0.1515994 0.1513076 0.1508096
## 0.75 0.1508129 0.1506464 0.1496506
## 1    0.1485745 0.1444121 0.1426281
## 1.25 0.1426281 0.1426281 0.1426281
## 1.5  0.1426281 0.1426281 0.1426281
## 1.75 0.1426281 0.1426281 0.1426281
## 2    0.1426281 0.1426281 0.1426281
## 2.5  0.1426281 0.1426281 0.1426281
## 3    0.1426281 0.1426281 0.1426281
## 3.5  0.1426281 0.1426281 0.1426281
## 4    0.1426281 0.1426281 0.1426281
## 4.5  0.1426281 0.1426281 0.1426281
## 5    0.1426281 0.1426281 0.1426281
##
##
## $Parameter
## [1] "all"
```

This Function might take some time to compute. Do not be alarmed. 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 step we will describe each of the functions.

Step 3 - Genome Wide Scoring

Computing Genome Wide metrics that will be used further down the line.

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

```
## 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

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

## 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:   0.8547021

## DNA Sequence Length: 3112514
## 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. SitesAboveThreshold

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

```
## Processing DNA Acccssibility
## Extracting Sites Above threshold
```

```
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
##
## PWM Scores at Sites higher than Threshold:
## GRangesList object of length 1:
## $eve
## GRanges object with 416 ranges and 1 metadata column:
##         seqnames             ranges strand |          PWMScore
##            <Rle>          <IRanges>  <Rle> |         <numeric>
##     [1]    chr2R [5860705, 5860712]      + | -1.84573024098586
##     [2]    chr2R [5860709, 5860716]      + | -4.96148500199546
##     [3]    chr2R [5860715, 5860722]      + |  8.81832070896316
##     [4]    chr2R [5860728, 5860735]      + |  4.24981127739825
##     [5]    chr2R [5860758, 5860765]      + | -5.25856937621247
##     ...      ...                ...    ... .               ...
##   [412]    chr2R [5876629, 5876636]      + |  5.76325435176529
##   [413]    chr2R [5876635, 5876642]      + | 0.824810948340001
##   [414]    chr2R [5876641, 5876648]      - |  -5.0584607351313
##   [415]    chr2R [5876666, 5876673]      + |  1.87745682827728
##   [416]    chr2R [5876684, 5876691]      + | -2.38839005613713
##
## -------
## 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:   0.8547021
## DNA Sequence Length: 3112514
## 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
##
## PWM Scores at Sites higher than Threshold:
## $`lambda = 1.5 & boundMolecules = 1000`
## GRangesList object of length 1:
## $eve
## GRanges object with 416 ranges and 2 metadata columns:
##       seqnames             ranges strand |          PWMScore
##          <Rle>          <IRanges>  <Rle> |         <numeric>
##   eve    chr2R [5860705, 5860712]      + | -1.84573024098586
##   eve    chr2R [5860709, 5860716]      + | -4.96148500199546
##   eve    chr2R [5860715, 5860722]      + |  8.81832070896316
##   eve    chr2R [5860728, 5860735]      + |  4.24981127739825
##   eve    chr2R [5860758, 5860765]      + | -5.25856937621247
##   ...      ...                ...    ... .               ...
##   eve    chr2R [5876629, 5876636]      + |  5.76325435176529
##   eve    chr2R [5876635, 5876642]      + | 0.824810948340001
##   eve    chr2R [5876641, 5876648]      - |  -5.0584607351313
##   eve    chr2R [5876666, 5876673]      + |  1.87745682827728
##   eve    chr2R [5876684, 5876691]      + | -2.38839005613713
##                Occupancy
##                  <numeric>
##   eve 0.0138657186001052
##   eve  0.013818354393958
##   eve 0.0758889551007343
##   eve 0.0169525472669577
##   eve 0.0138171354265231
##   ...                ...
##   eve 0.0223789395499536
##   eve 0.0141326917724132
##   eve  0.013817929667077
##   eve 0.0144591568695629
##   eve 0.0138492819301372
##
## -------
## 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:   0.8547021
## DNA Sequence Length: 3112514
## Strand Rule: max
## Strand: +-
```

This function will return a `genomicProfileParameters` object with an updated `AllSitesAboveThreshold`. Now the Occupancy values for each sites is 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 score from occupancy values.

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

```
## 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]      * | 0.0866275204260946
##   eve    chr2R [5860703, 5860713]      * | 0.0910690083746768
##   eve    chr2R [5860713, 5860723]      * | 0.0957382162799238
##   eve    chr2R [5860723, 5860733]      * | 0.0980474261454845
##   eve    chr2R [5860733, 5860743]      * | 0.0995892607500618
##   ...      ...                ...    ... .                ...
##   eve    chr2R [5876643, 5876653]      * | 0.0315681177094092
##   eve    chr2R [5876653, 5876663]      * | 0.0305804317246304
##   eve    chr2R [5876663, 5876673]      * | 0.0296692127477987
##   eve    chr2R [5876673, 5876683]      * | 0.0283988536820633
##   eve    chr2R [5876683, 5876693]      * |  0.027013825244469
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

This function will return a list of `GRangesList` with ChIP-seq like scores for every n base pairs (with n = stepSize, see bellow).

Step 6 - Model Accuracy

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

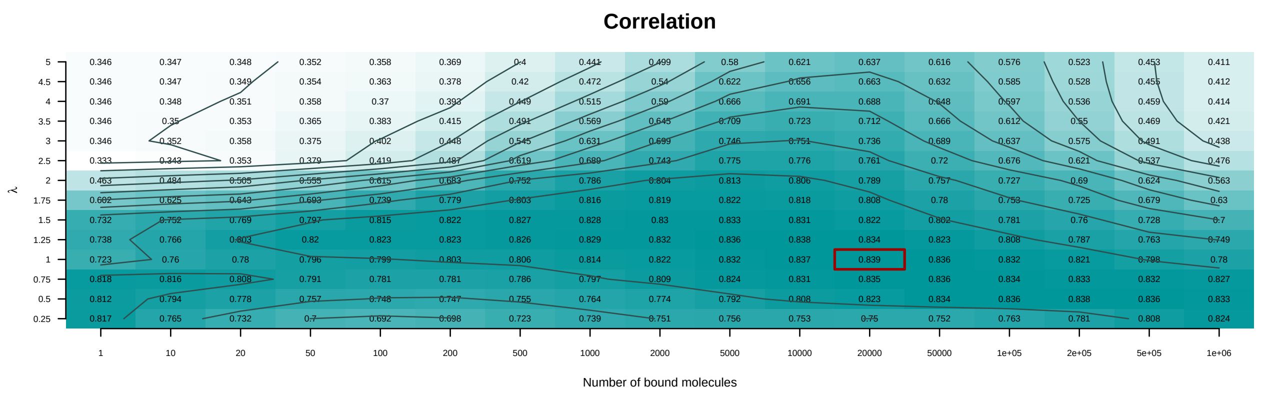
```
AccuracyEstimate <- profileAccuracyEstimate(LocusProfile = eveLocusChip,
    predictedProfile = chipProfile, occupancyProfileParameters = OPP)
AccuracyEstimate
```

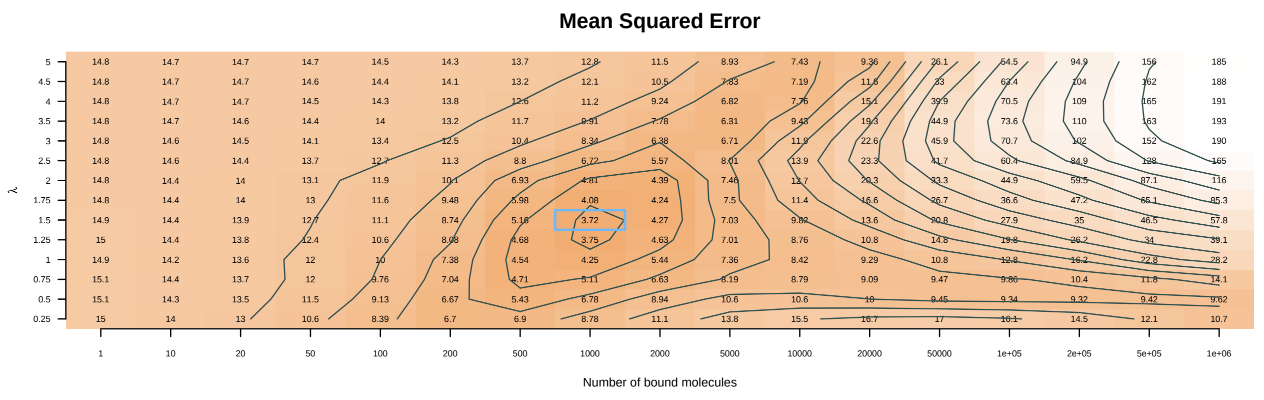
```
## $`lambda = 1.5 & boundMolecules = 1000`
## $`lambda = 1.5 & boundMolecules = 1000`$eve
##        Corr         MSE    meanCorr     meanMSE   meanTheta
## 0.828328636 0.003723466 0.828328636 3.723465521 0.222461745
```

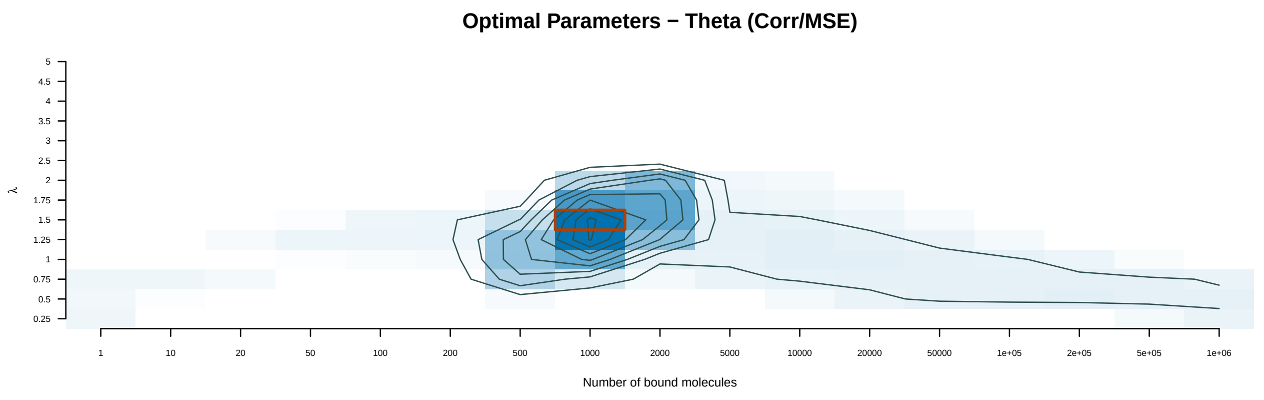
Step 7 - Plotting

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

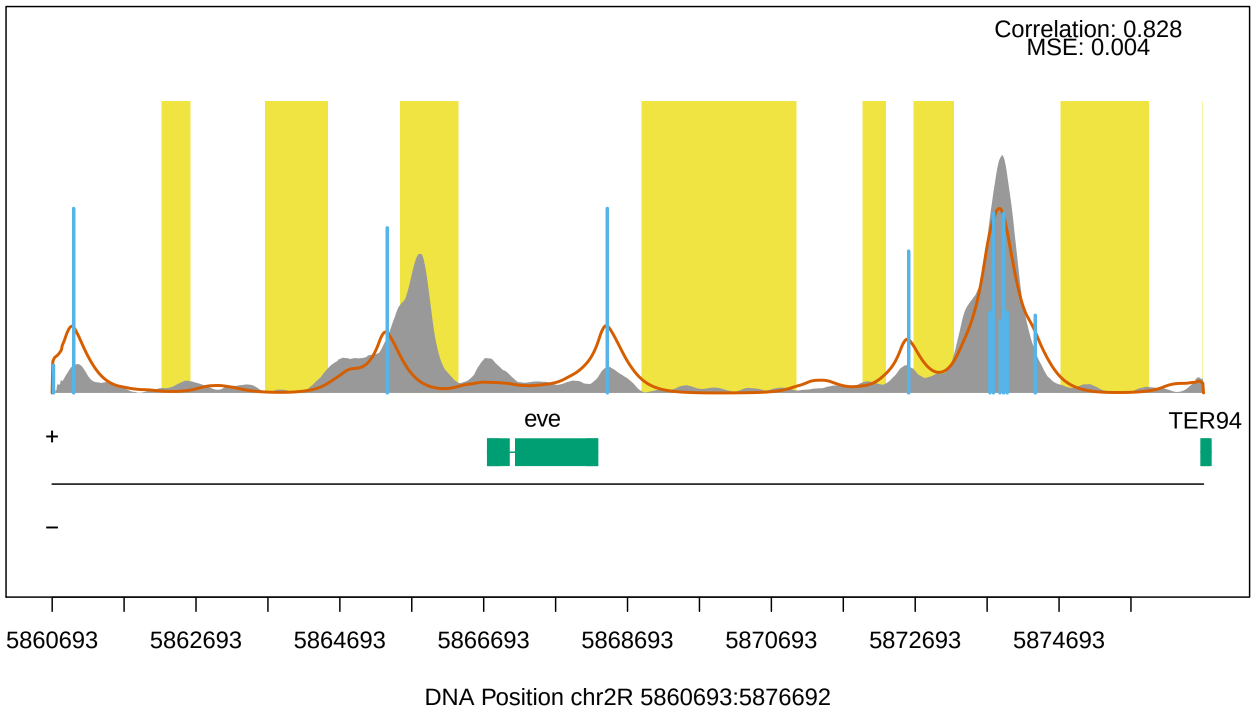
```
# Plotting Optimal heat maps
plotOptimalHeatMaps(optimalParam, parameter="all")
```







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



Work Flow - Full Guide

This section will described `ChIPanalyser` work flow. However in this section we will describe in detail data objects, parameters, and functions. Please refer to this section if in doubt to the use of any function or parameter.

Data objects - Genomic Profile Parameters

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

```
genomicProfileParameters(PWM, PFM, lambda, 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, lambda, 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)
# Suggested Minimal Build
GPP <- genomicProfileParameters(PFM=PFM, 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 two slots are absolutely 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`.

```
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
```

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

```
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
```

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. All other arguments are optional however we strongly recommend to tailor the values assigned `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` is may be inferred by using `computeOptimal`. Different values for `ScalingFactorPWM` will influence the fitness of the model. For more information, see `computeOptimal` and `profileAccuracyEstimate`.

```
ScalingFactorPWM(GPP)

ScalingFactorPWM(GPP) <- 0.5

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

- `PWMpseudocount`, a probability modifier. When computing a `PWM` from a `PFM`, it is possible that certain base pairs are completely absent from the Position Frequency Matrix. This absence will lead to odd results as part of this transformation requires a logarithmic transformation (at Position probability matrix step - a Matrix that describes the simple probability of a base pair being in that position of a binding motif given the PFM). *zeroes* will give minus infinities. In order to overcome this problem,a `PWMpseudocount` is introduced in the Position Probability Matrix. a `PWMpseudocount` of 1 (Default Value is 1) will then become a 0 after logarithmic transformation thus removing any mathematical discomforts.

```
PWMpseudocount(GPP)

PWMpseudocount(GPP) <- 1
```

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

```
BPFrequency(GPP)

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

```r
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. Choosing between *natural logarithm* or *log2* will depend on the *pseudocount* used. 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).

```r
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`.

```r
noOfSites(GPP)

noOfSites(GPP) <- 8
```

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

```r
PWMThreshold(GPP)

PWMThreshold(GPP) <- 0.7
```

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

```r
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 “-+”).

```r
whichstrand(GPP)

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

Genomic Parameters - The Updated ones

Some of the slots `genomicProfileParameters` should not be changed by user. Although it is still possible if absolutely required, we strongly advise against changing these slots. Certain Parameters are updated after a certain computation has been carried out. For example, `maxPWMScore` and `minPWMScore` are computed during the `computeGenomeWidePWMScore` function (see below) and represent both the highest and the lowest score of the given DNA sequence. These slots will be updated in the `genomicProfileParameters` object as one makes its way through the ChIPAnalyser work flow. Essentially, they are place holders for information required further down the work flow. Only slots that are of interest for the user are available for visualisation. If these slots have note been updated, the function will won't 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 value assigned to `lambda`. It is possible to access this slot using:

```
averageExpPWMScore(Occupancy)
```

```
## [1] 0.8547021
```

- `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] 3112514
```

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

```
NoAccess(Occupancy)
```

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

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

```
AllSitesAboveThreshold(Occupancy)
```

```
## $`lambda = 1.5 & boundMolecules = 1000`
## GRangesList object of length 1:
```

```
## $eve
## GRanges object with 416 ranges and 2 metadata columns:
##       seqnames             ranges strand |          PWMScore
##          <Rle>          <IRanges>  <Rle> |         <numeric>
##   eve    chr2R [5860705, 5860712]      + | -1.84573024098586
##   eve    chr2R [5860709, 5860716]      + | -4.96148500199546
##   eve    chr2R [5860715, 5860722]      + |  8.81832070896316
##   eve    chr2R [5860728, 5860735]      + |  4.24981127739825
##   eve    chr2R [5860758, 5860765]      + | -5.25856937621247
##   ...      ...                ...    ... .               ...
##   eve    chr2R [5876629, 5876636]      + |  5.76325435176529
##   eve    chr2R [5876635, 5876642]      + | 0.824810948340001
##   eve    chr2R [5876641, 5876648]      - |  -5.0584607351313
##   eve    chr2R [5876666, 5876673]      + |  1.87745682827728
##   eve    chr2R [5876684, 5876691]      + | -2.38839005613713
##                Occupancy
##                <numeric>
##   eve 0.0138657186001052
##   eve  0.013818354393958
##   eve 0.0758889551007343
##   eve 0.0169525472669577
##   eve 0.0138171354265231
##   ...                ...
##   eve 0.0223789395499536
##   eve 0.0141326917724132
##   eve  0.013817929667077
##   eve 0.0144591568695629
##   eve 0.0138492819301372
##
## -------
## seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

Data Objects - Occupancy Profile Parameters

`genomicProfileParameters` represent 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 tailored parameters in order to increase model fitness. It is especially the case here, as slot such as `maxSignal` are directly extracted from biological data (ChIP-seq data - see `computeChipProfile` and `profileAccuracyEstimate` for more information).

```
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 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.

```r
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 1000). In this model, occupancy is reliant on the number of bound molecules. The number of molecules will influence the fitness 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`.

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

- `backgroundSignal`, a numeric value representing the background Signal in real ChIP-seq data (Default is set at 0). It is strongly advised to set this parameter to the background Signal of the ChIP-seq data you will be using.

```r
backgroundSignal(OPP)

backgroundSignal(OPP) <- 0.02550997
```

- `maxSignal`, a numeric value representing the maximum signal in real ChIP-seq data (Default is set at 1). It is strongly advised to set this parameter to the maximum Signal of the ChIP-seq data you will be using.

```r
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). It is strongly advised to set this parameter to the average peak width of the ChIP-seq data you will be using.

```r
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). It is strongly advised to set this parameter to the SD peak width of the ChIP-seq data you will be using.

```r
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`.

```r
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 correaltion and MSE. Values below this threshold are discarded (replaced by correlation and MSE 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`.

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). The arguments of the function are the following :

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

Input Data - Genome Wide scoring

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

```
DNASequenceSet
```

```
##   A DNAStringSet instance of length 15
##         width seq                                         names
##   [1] 23011544 CGACAATGCACGACAGAGG...ATGAACCCCCCTTTCAAA chr2L
##  [2] 21146708 GACCCGCTAGGAGATGTTG...TTTGCATTCTAGGAATTC chr2R
##  [3] 24543557 TAGGGAGAAATATGATCGC...AACCAAGTTAATGTTCGG chr3L
##  [4] 27905053 GAATTCTCTCTTGTTGTAG...TTCGCATTCTAGGAATTC chr3R
##  [5]  1351857 GAATTCGCGTCCGCTTACC...CGATTTGAGATATATGAA chr4
##  ...      ... ...
## [11]  2555491 AACGAGGCCCATTTCATAC...ATGCCATTCGCTAGAAGT chr3LHet
## [12]  2517507 CCCTGTTTGCATCAGCGTT...TAAAAACAATTTGCTCCC chr3RHet
## [13]   204112 TAGATAGATAGATAGATAG...ATCGGAGTTAATGTTTGC chrXHet
## [14]   347038 AGGGTCACGTAATGCTGAT...TTGTTTTCCCCGGGATTG chrYHet
## [15] 29004656 ATTGAAAATGGATTGCATT...CAAGACCTTTCAAGACAA chrUextra
```

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

```
#Extracting DNAStringSet from BSgenome

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

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

```
GPP
```

```
## Object Class:genomicProfileParameters
##

##
## PWM:

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

##
## PFM:

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

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

## 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 optinal argument in `computeGenomeWidePWMScore`. If present **AND** `GenomeWide = FALSE`, 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
```

Finally, `verbose` will determine if progress messages should be printed in the console.

computeGenomeWidePWMScore

As an example of `computeGenomeWidePWMScore` usage:

```
# With DNAAccessibility

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

GenomeWide

# Without DNA accessibility
```

```r
GenomeWide <- computeGenomeWidePWMScore(DNASequenceSet = DNASequenceSet,
    genomicProfileParameters = GPP)
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`). The arguments of this functions are the following:

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

Input Data - Sites Above threshold

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

```r
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 imperative you name your loci of interest (not to be confused with `seqnames`). 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.

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

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

```r
# Names of Loci

names(eveLocus)
```

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

```r
# 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)
PWMScores

# Without DNA Accessibility

PWMScores <- computePWMScore(DNASequenceSet = DNASequenceSet,
    genomicProfileParameters = GenomeWide,
    setSequence = eveLocus)
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 you 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 `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 fitness. As a reminder, a `occupancyProfileParameters` object (previously created - see section **Data object - Occupancy profile Parameters**) should print on the screen as follows:

```
OPP
```

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

```
## Ploidy: 2

## boundMolecules:  1000

## backgroundSignal: 0.02550997
## maxSignal: 1.847
## chipMean: 200
## chipSd: 200
## 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 GRanges 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 you 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. The arguments of `computeChipProfile` are the following:

```
computeChipProfile( setSequence ,
    occupancy, occupancyProfileParameters = NULL, norm = TRUE,
    quick = TRUE, 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. Finally, `norm = TRUE` and `quick = TRUE` 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.

computeChipProfile

To generate a ChIP-seq like profile:

```r
chipProfile <- computeChipProfile(setSequence = eveLocus,
    occupancy = Occupancy,occupancyProfileParameters = OPP)
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 GenomicRanges object 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`).

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` provide a way to compare both profiles. The arguments for this function are the following:

```r
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:

```r
str(eveLocusChip)
```

```
## List of 1
##  $ eve: num [1:16000] 0.00755 0.00755 0.00755 0.00755 0.00755 ...
```

profileAccuracyEstimate

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

```r
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 fund the sweet spot between high correlation and low MSE. 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")
```

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

Input Data - Optimal Parameters

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

- `DNASequenceSet`, a DNAStringSet (or BSgenome) containing the sequences of the organism of interest.
- `genomicProfileParameters`, a `genomicProfileParameters` object containing at least a *Position Weight Matrix* or *Position Frequency Matrix*. All other slots will be computed internally.
- `LocusProfile`, a named list a 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`

The last argument, `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.

computeOptimal

As a example describing the usage of `compute optimal`

```r
optimalParam <- computeOptimal(DNASequenceSet = DNASequenceSet,
    genomicProfileParameters = GPP,
    LocusProfile = eveLocusChip,
    setSequence = eveLocus,
    DNAAccessibility = Access,
    occupancyProfileParameters = OPP,
    parameter = "all")
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 the case in mamy fields, data visualisation is a key aspect in any analysis. For this purpose, `ChIPAnalyser` offers two plotting functions: `plotOptimalHeatMaps` and `plotOccupancyProfile`.

Optimal Parameters

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

```r
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:

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

This return a heat map resembling the following:

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. `plotOccupancyProfile` takes care of plotting with the following arguments:

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

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 GRangesList 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.

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

```
geneRef
```

```
## $exon
## GRanges object with 26713 ranges and 0 metadata columns:
##           seqnames               ranges strand
##              <Rle>            <IRanges>  <Rle>
##   CG17683    chr2R       [18442, 18629]      +
##   CG17683    chr2R       [18681, 18773]      +
##   CG17683    chr2R       [18827, 19484]      +
##   CG17683    chr2R       [19542, 20468]      +
##   CG17683    chr2R       [18442, 18629]      +
##       ...      ...                  ...    ...
##   CG33680    chr2R [21137781, 21137839]      -
##   CG30428    chr2R [21140837, 21140963]      +
##   CG30428    chr2R [21141104, 21141284]      +
##   CG30428    chr2R [21141343, 21141601]      +
##   CG30428    chr2R [21141651, 21142371]      +
##   -------
##   seqinfo: 14 sequences from an unspecified genome; no seqlengths
##
## $intron
## GRanges object with 22058 ranges and 0 metadata columns:
##           seqnames               ranges strand
##              <Rle>            <IRanges>  <Rle>
##   CG17683    chr2R       [18630, 18680]      +
##   CG17683    chr2R       [18774, 18826]      +
##   CG17683    chr2R       [19485, 19541]      +
##   CG17683    chr2R       [18630, 18692]      +
##   CG17683    chr2R       [18774, 18826]      +
##       ...      ...                  ...    ...
##   CG33680    chr2R [21137114, 21137174]      -
##   CG33680    chr2R [21137423, 21137780]      -
##   CG30428    chr2R [21140964, 21141103]      +
##   CG30428    chr2R [21141285, 21141342]      +
##   CG30428    chr2R [21141602, 21141650]      +
##   -------
##   seqinfo: 13 sequences from an unspecified genome; no seqlengths
##
## $`5UTR`
## GRanges object with 6029 ranges and 0 metadata columns:
##           seqnames               ranges strand
##              <Rle>            <IRanges>  <Rle>
##   CG17683    chr2R       [18442, 18566]      +
##   CG17683    chr2R       [18442, 18566]      +
##   CG17683    chr2R       [18487, 18629]      +
##   CG17683    chr2R       [18681, 18811]      +
##   CG17683    chr2R       [18498, 18773]      +
##       ...      ...                  ...    ...
##    CG9380    chr2R [21076340, 21076360]      -
##    CG9380    chr2R [21076340, 21076360]      -
##        Kr    chr2R [21114138, 21114474]      +
##   CG30429    chr2R [21133990, 21134051]      +
##   CG30428    chr2R [21140837, 21140961]      +
##   -------
##   seqinfo: 13 sequences from an unspecified genome; no seqlengths
##
## $`3UTR`
## GRanges object with 4556 ranges and 0 metadata columns:
##           seqnames               ranges strand
##              <Rle>            <IRanges>  <Rle>
##   CG17683    chr2R       [20162, 20468]      +
##   CG17683    chr2R       [20162, 20468]      +
##   CG17683    chr2R       [20162, 20468]      +
##   CG17683    chr2R       [20162, 20468]      +
##   CG17683    chr2R       [20162, 20468]      +
##       ...      ...                  ...    ...
##    CG9380    chr2R [21072649, 21072809]      -
##        Kr    chr2R [21116357, 21117057]      +
##   CG30429    chr2R [21135028, 21135109]      +
##   CG33680    chr2R [21136529, 21136529]      -
##   CG30428    chr2R [21142001, 21142371]      +
##   -------
##   seqinfo: 13 sequences 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,
    profileAccuracy = AccuracyEstimate[[1]][[1]],
    chipProfile = eveLocusChip[[1]],
    occupancy = AllSitesAboveThreshold(Occupancy)[[1]][[1]],
    DNAAccessibility = Access,
    occupancyProfileParameters = OPP,
    geneRef =geneRef)
```

And the resulting plot should look similar to the following:

Session Information

```
sessionInfo()
```

```
## R version 3.3.2 (2016-10-31)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 16.04.3 LTS
##
## locale:
##  [1] LC_CTYPE=en_GB.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_GB.UTF-8        LC_COLLATE=en_GB.UTF-8
##  [5] LC_MONETARY=en_GB.UTF-8    LC_MESSAGES=en_GB.UTF-8
##  [7] LC_PAPER=en_GB.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_GB.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_0.99.0
##  [3] BSgenome_1.42.0
##  [4] rtracklayer_1.34.2
##  [5] Biostrings_2.42.1
##  [6] XVector_0.14.1
##  [7] GenomicRanges_1.26.4
##  [8] GenomeInfoDb_1.10.3
##  [9] IRanges_2.8.2
## [10] S4Vectors_0.12.2
## [11] BiocGenerics_0.20.0
```

```
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.12               knitr_1.17
##  [3] magrittr_1.5               GenomicAlignments_1.10.1
##  [5] zlibbioc_1.20.0            BiocParallel_1.8.2
##  [7] lattice_0.20-35            stringr_1.2.0
##  [9] tools_3.3.2                grid_3.3.2
## [11] SummarizedExperiment_1.4.0 Biobase_2.34.0
## [13] htmltools_0.3.6            yaml_2.1.14
## [15] rprojroot_1.2              digest_0.6.12
## [17] Matrix_1.2-10              bitops_1.0-6
## [19] RCurl_1.95-4.8             evaluate_0.10.1
## [21] rmarkdown_1.6              stringi_1.1.5
## [23] backports_1.1.0            Rsamtools_1.26.2
## [25] XML_3.98-1.9
```

References

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