Package ‘ChIPanalyser’

May 3, 2024

Type Package
Title ChIPanalyser: Predicting Transcription Factor Binding Sites
Version 1.26.0
Date 2017-09-01
Author Patrick C.N.Martin & Nicolae Radu Zabet
Maintainer Patrick C.N. Martin <pcnmartin@gmail.com>
Description ChIPanalyser is a package to predict and understand TF binding by utilizing a statistical thermodynamic model. The model incorporates 4 main factors thought to drive TF binding: Chromatin State, Binding energy, Number of bound molecules and a scaling factor modulating TF binding affinity. Taken together, ChIPanalyser produces ChIP-like profiles that closely mimic the patterns seen in real ChIP-seq data.
License GPL-3
Depends R (>= 3.5.0), GenomicRanges, Biostrings, BSgenome, RcppRoll, parallel
Imports methods, IRanges,
S4Vectors, grDevices, graphics, stats, utils, rtracklayer, ROCR, BiocManager, GenomeInfoDb, RColorBrewer
Suggests BSgenome.Dmelanogaster.UCSC.dm6, knitr, RUnit, BiocGenerics
Encoding UTF-8
LazyData true
Contents

biocViews Software, BiologicalQuestion, WorkflowStep, Transcription, Sequencing, ChipOnChip, Coverage, Alignment, ChIPSeq, SequenceMatching, DataImport, PeakDetection

VignetteBuilder knitr
RoxygenNote 7.2.1

git_url https://git.bioconductor.org/packages/ChIPanalyser

Repository Bioconductor 3.19
Date/Publication 2024-05-03

ChIPanalyser-package ........................................... 5
averageExpPWMScore ........................................... 6
averageExpPWMScore-methods .................................. 7
backgroundSignal .............................................. 8
backgroundSignal-methods ..................................... 9
backgroundSignal<- ............................................ 9
backgroundSignal<-methods ................................... 10
boundMolecules ................................................. 11
boundMolecules-methods ....................................... 12
boundMolecules<- ............................................. 12
boundMolecules<-methods ..................................... 13
BPFrequency ..................................................... 13
BPFrequency-methods .......................................... 14
BPFrequency<- .................................................. 15
BPFrequency<-methods ......................................... 16
ChIPanalyserData ............................................... 16
chipMean ......................................................... 17
chipMean-methods ............................................... 18
chipMean<- ....................................................... 19
chipMean<-methods ............................................. 20
ChIPScore-class .................................................. 20
chipSd ............................................................. 22
chipSd-methods .................................................. 23
chipSd<- .......................................................... 23
chipSd<-methods ................................................ 24
chipSmooth ........................................................ 24
chipSmooth-methods ............................................ 25
chipSmooth<- ..................................................... 25
chipSmooth<-methods ........................................... 26
computeChIPProfile .............................................. 27
computeGenomeWideScores .................................... 29
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>computeOccupancy</td>
<td>30</td>
</tr>
<tr>
<td>computeOptimal</td>
<td>32</td>
</tr>
<tr>
<td>computePWMScore</td>
<td>34</td>
</tr>
<tr>
<td>DNASequenceLength</td>
<td>36</td>
</tr>
<tr>
<td>DNASequenceLength-methods</td>
<td>37</td>
</tr>
<tr>
<td>drop</td>
<td>38</td>
</tr>
<tr>
<td>drop-methods</td>
<td>39</td>
</tr>
<tr>
<td>evolve</td>
<td>39</td>
</tr>
<tr>
<td>generateStartingPopulation</td>
<td>41</td>
</tr>
<tr>
<td>genomicProfiles</td>
<td>42</td>
</tr>
<tr>
<td>genomicProfiles-class</td>
<td>43</td>
</tr>
<tr>
<td>genomicProfilesInternal-class</td>
<td>46</td>
</tr>
<tr>
<td>getHighestFitnessSolutions</td>
<td>48</td>
</tr>
<tr>
<td>getTestingData</td>
<td>49</td>
</tr>
<tr>
<td>getTrainingData</td>
<td>49</td>
</tr>
<tr>
<td>GRList-class</td>
<td>50</td>
</tr>
<tr>
<td>initialize-methods</td>
<td>51</td>
</tr>
<tr>
<td>lambdaPWM</td>
<td>51</td>
</tr>
<tr>
<td>lambdaPWM-methods</td>
<td>52</td>
</tr>
<tr>
<td>lambdaPWM&lt;-</td>
<td>52</td>
</tr>
<tr>
<td>lambdaPWM&lt;-methods</td>
<td>53</td>
</tr>
<tr>
<td>loci</td>
<td>53</td>
</tr>
<tr>
<td>loci-class</td>
<td>54</td>
</tr>
<tr>
<td>loci-methods</td>
<td>55</td>
</tr>
<tr>
<td>lociWidth</td>
<td>55</td>
</tr>
<tr>
<td>lociWidth-methods</td>
<td>56</td>
</tr>
<tr>
<td>lociWidth&lt;-</td>
<td>57</td>
</tr>
<tr>
<td>lociWidth&lt;-methods</td>
<td>58</td>
</tr>
<tr>
<td>maxPWMScore</td>
<td>58</td>
</tr>
<tr>
<td>maxPWMScore-methods</td>
<td>59</td>
</tr>
<tr>
<td>maxSignal</td>
<td>59</td>
</tr>
<tr>
<td>maxSignal-methods</td>
<td>60</td>
</tr>
<tr>
<td>maxSignal&lt;-</td>
<td>61</td>
</tr>
<tr>
<td>maxSignal&lt;-methods</td>
<td>62</td>
</tr>
<tr>
<td>minPWMScore</td>
<td>62</td>
</tr>
<tr>
<td>minPWMScore-methods</td>
<td>63</td>
</tr>
<tr>
<td>naturalLog</td>
<td>63</td>
</tr>
<tr>
<td>naturalLog-methods</td>
<td>64</td>
</tr>
<tr>
<td>naturalLog&lt;-</td>
<td>65</td>
</tr>
<tr>
<td>naturalLog&lt;-methods</td>
<td>66</td>
</tr>
<tr>
<td>noiseFilter</td>
<td>66</td>
</tr>
<tr>
<td>noiseFilter-methods</td>
<td>67</td>
</tr>
<tr>
<td>noiseFilter&lt;-</td>
<td>67</td>
</tr>
<tr>
<td>noiseFilter&lt;-methods</td>
<td>68</td>
</tr>
<tr>
<td>noOfSites</td>
<td>68</td>
</tr>
<tr>
<td>noOfSites-methods</td>
<td>69</td>
</tr>
<tr>
<td>noOfSites&lt;-</td>
<td>70</td>
</tr>
<tr>
<td>noOfSites&lt;-methods</td>
<td>71</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>nos-class</td>
<td>71</td>
</tr>
<tr>
<td>parameterOptions</td>
<td>72</td>
</tr>
<tr>
<td>parameterOptions-class</td>
<td>74</td>
</tr>
<tr>
<td>PFMFormat</td>
<td>77</td>
</tr>
<tr>
<td>PFMFormat-methods</td>
<td>78</td>
</tr>
<tr>
<td>PFMFormat&lt;-</td>
<td>78</td>
</tr>
<tr>
<td>PFMFormat&lt;-methods</td>
<td>79</td>
</tr>
<tr>
<td>ploidy</td>
<td>79</td>
</tr>
<tr>
<td>ploidy-methods</td>
<td>80</td>
</tr>
<tr>
<td>ploidy&lt;-</td>
<td>81</td>
</tr>
<tr>
<td>ploidy&lt;-methods</td>
<td>82</td>
</tr>
<tr>
<td>plotOccupancyProfile</td>
<td>82</td>
</tr>
<tr>
<td>plotOptimalHeatMaps</td>
<td>84</td>
</tr>
<tr>
<td>PositionFrequencyMatrix</td>
<td>86</td>
</tr>
<tr>
<td>PositionFrequencyMatrix-methods</td>
<td>87</td>
</tr>
<tr>
<td>PositionFrequencyMatrix&lt;-</td>
<td>87</td>
</tr>
<tr>
<td>PositionFrequencyMatrix&lt;-methods</td>
<td>88</td>
</tr>
<tr>
<td>PositionWeightMatrix</td>
<td>89</td>
</tr>
<tr>
<td>PositionWeightMatrix-methods</td>
<td>90</td>
</tr>
<tr>
<td>PositionWeightMatrix&lt;-</td>
<td>90</td>
</tr>
<tr>
<td>PositionWeightMatrix&lt;-methods</td>
<td>91</td>
</tr>
<tr>
<td>processingChIP</td>
<td>91</td>
</tr>
<tr>
<td>profileAccuracyEstimate</td>
<td>93</td>
</tr>
<tr>
<td>profiles-methods</td>
<td>95</td>
</tr>
<tr>
<td>PWMpseudocount</td>
<td>95</td>
</tr>
<tr>
<td>PWMpseudocount-methods</td>
<td>96</td>
</tr>
<tr>
<td>PWMpseudocount&lt;-</td>
<td>96</td>
</tr>
<tr>
<td>PWMpseudocount&lt;-methods</td>
<td>97</td>
</tr>
<tr>
<td>PWMThreshold</td>
<td>98</td>
</tr>
<tr>
<td>PWMThreshold-methods</td>
<td>99</td>
</tr>
<tr>
<td>PWMThreshold&lt;-</td>
<td>99</td>
</tr>
<tr>
<td>removeBackground</td>
<td>100</td>
</tr>
<tr>
<td>removeBackground-methods</td>
<td>100</td>
</tr>
<tr>
<td>removeBackground&lt;-</td>
<td>101</td>
</tr>
<tr>
<td>removeBackground&lt;-methods</td>
<td>101</td>
</tr>
<tr>
<td>scores</td>
<td>102</td>
</tr>
<tr>
<td>scores-methods</td>
<td>103</td>
</tr>
<tr>
<td>searchSites</td>
<td>104</td>
</tr>
<tr>
<td>setChromatinStates</td>
<td>106</td>
</tr>
<tr>
<td>show-methods</td>
<td>107</td>
</tr>
<tr>
<td>singleRun</td>
<td>107</td>
</tr>
<tr>
<td>splitData</td>
<td>108</td>
</tr>
<tr>
<td>stepSize</td>
<td>109</td>
</tr>
<tr>
<td>stepSize-methods</td>
<td>110</td>
</tr>
<tr>
<td>stepSize&lt;-</td>
<td>110</td>
</tr>
<tr>
<td>stepSize&lt;-methods</td>
<td>111</td>
</tr>
<tr>
<td>strandRule</td>
<td>111</td>
</tr>
</tbody>
</table>
ChIPanalyser-package

Description

ChIPanalyser is a package to predict and understand TF binding by utilizing a statistical thermodynamic model. The model incorporates 4 main factors thought to drive TF binding: Chromatin State, Binding energy, Number of bound molecules and a scaling factor modulating TF binding affinity. Taken together, ChIPanalyser produces ChIP-like profiles that closely mimic the patterns seen in real ChIP-seq data.

Details

The DESCRIPTION file: This package was not yet installed at build time.

Index: This package was not yet installed at build time.

Author(s)

Patrick C.N. Martin <pm16057@essex.ac.uk>
And
Nicolae Radu Zabet <nzabet@essex.ac.uk>
Maintainer: Patrick C.N. Martin <pcnmartin@gmail.com>

References


Examples

#Data extraction
data(ChIPanalyserData)
# path to Position Frequency Matrix
PFM <- file.path(system.file("extdata",package="ChIPanalyser"),"BEAF-32.pfm")
#As an example of genome, this example will run on the Drosophila genome
averageExpPWMScore

Accessor for averageExpPWMScore slot in a genomicProfiles object.

Description

Extract or Access averageExpPWMScore slot in a genomicProfiles

Usage

averageExpPWMScore(object)

Arguments

object object is a genomicProfiles
Details

As a general rule, \texttt{averageExpPWMScore} is computed and updated internally by \texttt{computeGenomeWideScores}. Ideally, this slot should not be updated by user. The \texttt{averageExpPWMScore} is the sum of the exponential of every PWM score for a given DNA sequence and divided by the length of the said DNA sequence (\texttt{DNASequenceLength}). This can either be the full length sequence or only the accessible sequence (see \texttt{computeGenomeWideScores}).

Value

Returns the \texttt{averageExpPWMScore} of a \texttt{genomicProfiles} when computed.

Author(s)

Patrick C.N. Martin <pcnmartin@gmail.com>

References


Examples

```r
# Accessing Data
data(ChIPanalyserData)
# path to Position Frequency Matrix
PFM <- file.path(system.file("extdata",package="ChIPanalyser"),"BEAF-32.pfm")
# Building genomicProfiles object
GPP <- genomicProfiles(PFM=PFM,PFMFormat="JASPAR")
# Extracting AllSitesAboveThreshold slot
averageExpPWMScore(GPP)

## Note this slot is now empty as nothing has yet been computed
```
backgroundSignal

Accessor method for the backgroundSignal slot in a parameterOptions object.

Description

Extract or access the backgroundSignal slot in a parameterOptions object.

Usage

backgroundSignal(object)

Arguments

object object is an parameterOptions

Details

Default Value: 0

When computing computeOccupancy, a ChIP-seq background signal is used to scale Occupancy by considering both a backgroundSignal and a maxSignal. The backgroundSignal is also used to normalise occupancies against maxOccupancy. The backgroundSignal usually comes from experimental data and is provided by user. As a general rule, if ChIP-seq data is available and will be used in computeChIPProfile, profileAccuracyEstimate or plotOccupancyProfile, it is advised to use the backgroundSignal from this data. We strongly encourage to set values when building a parameterOptions object.

Value

Returns a backgroundSignal of a parameterOptions object.

Author(s)

Patrick C.N. Martin <pcnmartin@gmail.com>

References

Examples

```r
# Building occupancyProfileParameters object
OPP <- parameterOptions()
# Viewing single value in object
backgroundSignal(OPP)
```

Description

Methods for function `backgroundSignal`.

Methods:

```r
signature(object = "parameterOptions")
```

Description

Setter method for `backgroundSignal` slot in a `parameterOptions`.

Usage

```r
backgroundSignal(object) <- value
```

Arguments

- `object`: object is an `parameterOptions` object.
- `value`: value is the value to be assigned to the `backgroundSignal` slot in `parameterOptions`. `backgroundSignal` should be a positive value. Default value is 0.

Details

Default value: 0. When computing `computeOccupancy`, a ChIP-seq background signal is used to scale Occupancy by considering both a `backgroundSignal` and a `maxSignal`. The `backgroundSignal` is also used to normalise occupancies to maxOccupancy. The `backgroundSignal` usually comes from experimental data and is provided by user. As a general rule, if ChIP-seq data is available and will be used in `computeChIPProfile`, `profileAccuracyEstimate` or `plotOccupancyProfile`, it is advised to use the `backgroundSignal` from this data. We strongly encourage to set values when building a `parameterOptions` object.
Returns a `parameterOptions` object with a new value assigned to the `backgroundSignal` slot.

**Author(s)**

Patrick C.N. Martin <pcnmartin@gmail.com>

**References**


**Examples**

```r
# Building occupancyProfileParameters object
OPP <- parameterOptions()
# Setting new value for backgroundSignal
backgroundSignal(OPP) <- 0.2
# Viewing whole object with new updated value
OPP
# Viewing single value in object
backgroundSignal(OPP)
```

```r

backgroundSignal<-methods

~ Methods for Function backgroundSignal<- ~
```
boundMolecules

Accessor methods for boundMolecules slot in parameterOptions object.

Description

Extract or Access boundMolecules slot in parameterOptions object.

Usage

boundMolecules(object)

Arguments

object object is a parameterOptions object.

Details

Default value: 1000

When computing occupancy (computeOccupancy), a value for the number of bound Molecules to DNA is needed. This value can be updated and set in a parameterOptions object. If the number of molecules is unknown, it is possible to infer this value with computeOptimal. We strongly encourage to set values when building a parameterOptions object.

Value

Returns boundMolecules slot in parameterOptions object.

Author(s)

Patrick C.N. Martin <pcnmartin@gmail.com>

References


Examples

# Building parameterOptions object
OPP <- parameterOptions()
#Checking single value by slot accessor
boundMolecules(OPP)
boundMolecules<-  

~~ Methods for Function boundMolecules ~~

**Description**

~~ Methods for function boundMolecules ~~

**Methods:**

signature(object = "parameterOptions")

---

**boundMolecules<-  Setter method for the boundMolecules slot in a parameterOptions object.**

**Description**

Setter method for the boundMolecules slot in a parameterOptions object.

**Usage**

boundMolecules(object)<-value

**Arguments**

- **object**  
  object is a parameterOptions object.

- **value**  
  value is a positive integer or vector of positive integers describing the number of molecules bound to DNA. Default value is 1000.

**Details**

Default value: 1000 When computing occupancy (computeOccupancy), a value for the number of bound Molecules to DNA is needed. This value can be updated and set in a parameterOptions object. If the number of molecules is unknown, it is possible to infer this value with computeOptimal. We strongly encourage to set values when building a parameterOptions object.

**Value**

Returns a parameterOptions object with an updated value for boundMolecules.

**Author(s)**

Patrick C.N. Martin <pcnmartin@gmail.com>
References


Examples

# Building parameterOptions object
OPP <- parameterOptions()
# Setting new boundMolecules value in OPP
boundMolecules(OPP) <- 5000
# Checking value in whole object
OPP
# Checking single value by slot accessor
boundMolecules(OPP)

boundMolecules<-methods

~~ Methods for Function boundMolecules<- ~~

Description

~~ Methods for function boundMolecules<- ~~

Methods:

signature(object = "parameterOptions", value = "vector")

BPFrequency

Accessor method for BPFrequency slot in a genomicProfiles object.

Description

Extract or Access BPFrequency slot in a genomicProfiles object.

Usage

BPFrequency(object)

Arguments

object object is a genomicProfiles
Details

Default value is \(c(0.25, 0.25, 0.25, 0.25)\) When generating a Position Weight Matrix from a Position Frequency Matrix, the probability of occurrence of each base pair (Base Pair Frequency) is necessary (as originally described by Gary Stormo). It is possible to set custom values for \(\text{BPFrequency}\) with a vector of length 4 containing the probability of occurrence of each base pair (A,C,G,T) in order. If Base pair frequency is unknown, \(\text{BPFrequency}\) will compute base pair frequency from a DNA sequence. The nature of this sequence can be a BSgenome or a DNAStringSet. In order to decrease run time, it is advised to use DNAStringSet.

Value

Returns BPFrequency slot in genomicProfiles object.

Author(s)

Patrick C.N. Martin <pcnmartin@gmail.com>

References


Examples

data(ChIPanalyserData)
# path to Position Frequency Matrix
PFM <- file.path(system.file("extdata",package="ChIPanalyser"),"BEAF-32.pfm")
# Building genomicProfiles object
GPP <- genomicProfiles(PFM=PFM,PFMFormat="JASPAR")
# Extracting BPFrequency slot
BPFrequency(GPP)

BPFrequency-methods ~~ Methods for Function BPFrequency ~~

Description

~~ Methods for function BPFrequency ~~

Methods:

signature(object = "genomicProfilesInternal")
**BPFrequency**<-  

*Setter method for BPFrequency slot in a genomicProfiles object.*

**Description**

Setter method for BPFrequency slot in a genomicProfiles object. If base pair frequency is unknown, BPFrequency will compute base pair frequency from a DNA sequence.

**Usage**

BPFrequency(object)<-value

**Arguments**

- **object** object is a genomicProfiles object.
- **value** value can three different objects:
  - A vector of length 4 containing the probability of occurrence of each base pair (A,C,G,T) in order. Default value is c(0.25,0.25,0.25,0.25).
  - A BSgenome of the organism of interest. The base pair frequency will automatically be computed and updated in genomicProfiles.
  - A DNAStringSet of the organism of interest. The base pair frequency will automatically be computed and updated in genomicProfiles (Prefered method).

**Details**

Default value is c(0.25,0.25,0.25,0.25) When generating a Position Weight Matrix from a Position Frequency Matrix, the probability of occurrence of each base pair (Base Pair Frequency) is necessary (as originally described by Gary Stormo). It is possible to set custom values for BPFrequency with a vector of length 4 containing the probability of occurrence of each base pair (A,C,G,T) in order. If Base pair frequency is unknown, BPFrequency will compute base pair frequency from a DNA sequence when building a genomicProfiles object. The nature of this sequence can be a BSgenome object or a DNAStringSet. In order to decrease run time, it is advised to use DNAStringSet.

**Value**

Returns a genomicProfiles object with an updated value for BPFrequency.

**Author(s)**

Patrick C.N. Martin <pcnmartin@gmail.com>

**References**

Examples

data(ChIPanalyserData)
# path to Position Frequency Matrix
PFM <- file.path(system.file("extdata",package="ChIPanalyser"),"BEAF-32.pfm")
#As an example of genome, this example will run on the Drosophila genome

if(!require("BSgenome.Dmelanogaster.UCSC.dm6", character.only = TRUE)){
  if (!requireNamespace("BiocManager", quietly=TRUE))
    install.packages("BiocManager")
  BiocManager::install("BSgenome.Dmelanogaster.UCSC.dm6")
}
library(BSgenome.Dmelanogaster.UCSC.dm6)
DNASequenceSet <- getSeq(BSgenome.Dmelanogaster.UCSC.dm6)
# Building genomicProfiles object
GPP <- genomicProfiles(PFM=PFM,PFMFormat = "JASPAR", BPFrequency=DNASequenceSet)
# Updating BPFrequency
## !! Note!! BPFrequency is used to compute PWM from PFM
## IF updated after building GPP, then it will not influence PWM
## Advised to build with BPFrequency directly
BPFrequency(GPP) <- DNASequenceSet
BPFrequency(GPP) <- c(0.25,0.25,0.25,0.25)

Description

~~ Methods for function BPFrequency<- ~~

Methods:

signature(object = "genomicProfilesInternal", value = "DNAStringSet")
signature(object = "genomicProfilesInternal", value = "vector")

ChIPanalyserData

Description

ChIPanalyserData is derived from real biological data. The source organism is *Drosophila melanogaster*. The data can be described as genomic data as it contains DNA sequences, loci, genetic information, DNA accessibility data and ChIP-seq data.

Usage

data(ChIPanalyserData)
Format

1. Accessis `GRanges` containing DNA Accessibility data for the sequences described above.
2. `csis` `GRanges` containing Chromatin State data for the sequences described above.
3. `topi` `GRanges` containing a locus of interest. In this case *eve strip Locus* on chromosome 2R in *Drosophila melanogaster*.
4. `chipis` `GRanges` containing ChIP score of the *eve strip* locus in *Drosophila melanogaster*.
5. `geneRefis` `GRanges` containing UCSC gene reference information.

Value

Returns a set of Rdata objects as described above.

Source

Transcription Factor PFM: Berkeley Drosophila Transcription Network Project (bdtnp.lbl.gov)

References


Examples

data(ChIPanalyserData)

---

**chipMean**

Accessor method for chipMean slot in a parameterOptions object.

Description

Accessor method for chipMean slot in a parameterOptions object.

Usage

chipMean(object)

Arguments

object object is a parameterOptions
Details

Default value: 150 When computing ChIP-seq like profiles (\texttt{computeChIPProfile}), the occupancy values given by \texttt{computeOccupancy} are transformed into ChIP-seq like profiles. The average size of a ChIP-seq peak was described by Kaplan (Kaplan et al., 2011). It is advised to use the average width of ChIP peaks from actual ChIP-seq data. We strongly encourage to set values when building a \texttt{parameterOptions} object.

Value

Returns \texttt{chipMean} slot from a \texttt{parameterOptions} object.

Author(s)

Patrick C.N. Martin $<$pcnmartin@gmail.com$>$

References


Examples

```r
# Building parameterOptions object
OPP <- parameterOptions()
#Accessing chipMean slot in OPP
chipMean(OPP)
```

chipMean-methods

Description

~~ Methods for function chipMean ~~

Methods:

\texttt{chipMean(object)}
Access methods for chipMean slot in parameterOptions object.

Description

Access methods for chipMean slot in parameterOptions object.

Usage

chipMean(object) <- value

Arguments

object object is a parameterOptions object.
value value is a positive numeric value that will be assigned to the chipMean slot.
chipMean describes the average size of a ChIP-seq peak in base pairs.

Details

Default value: 150 When computing ChIP-seq like profiles (computeChIPProfile, the occupancy values given by computeOccupancy are transformed into ChIP-seq like profiles. The average size of a ChIP-seq peak was described by Kaplan (Kaplan et al., 2011). It is advised to use the average width of ChIP peaks from actual ChIP-seq data. We strongly encourage to set values when building a parameterOptions object.

Value

Returns a parameterOptions object with an updated value for chipMean slot.

Author(s)

Patrick C.N. Martin <pcnmartin@gmail.com>

References


Examples

# Building parameterOptions object
OPP <- parameterOptions()
# Setting new value for slot
chipMean(OPP) <- 250
Description

ChIPScore is the result of the processingChIP function. This object contains the extracted ChIP Score from ChIP data, the loci of interest and optional parameters associated to ChIPanlyser. The loci of interest will either be user provided or the top n regions as defined by the reduce argument in processingChIP. This object has the sole purpose of aiding the storage and parsing of data and parameters.

Objects from the Class

Object of this class are created internally and will be parsed to other objects as is.

Slots

scores: Object of class "list" List of extracted ChIP scores
loci: Object of class "loci" GRanges containing loci of interest
ploidy: Object of class "numeric" Ploidy level of the organism
boundMolecules: Object of class "vector" Number of Bound molecules to the DNA
backgroundSignal: Object of class "numeric" ChIP background signal (average ChIP score)
maxSignal: Object of class "numeric" max ChIP signal
lociWidth: Object of class "numeric" Width of loci if reduce is used and no loci are provided
chipMean: Object of class "numeric" Average ChIP peak width
chipSd: Object of class "numeric" Standard Deviation of ChIP peak width
chipSmooth: Object of class "vector" Smoothing window width for ChIP score
stepSize: Object of class "numeric" Defining resolution size of ChIP like profiles (10bp = signal will be only considered every 10bp)
removeBackground: Object of class "numeric" Signal Threshold to be removed. Default removes all negative scores
noiseFilter: Object of class "character" Type of noise filter to be used on ChIP data.
PWMThreshold: Object of class "numeric" Threshold of PWM scores that will be selected
strandRule: Object of class "character" Rule to compute strand score (max, mean or sum)
whichstrand: Object of class "character" Which strand should be used to compute PWM scores.
lambdaPWM: Object of class "vector" Lambda value - Scaling factor to the PWM
naturalLog: Object of class "logical" PFM to PWM conversion log transform (natural log or log2)
noOfSites: Object of class "nos" Number of Sites in the PWM that should be used to compute PWM scores.
PWMpseudocount: Object of class "numeric" PWM pseudocount value for PFM to PWM conversion.
paramTag: Object of class "character" Internal Tag - Code progression

Extends
Class "parameterOptions", directly.

Methods
.loci<- signature(object = "ChIPScore", value = "loci"): ...
.scores<- signature(object = "ChIPScore", value = "list"): ...
initialize signature(.Object = "ChIPScore"): ...
loci signature(object = "ChIPScore"): ...
scores signature(object = "ChIPScore"): ...
show signature(object = "ChIPScore"): ...

Author(s)
Patrick C.N. Martin

References

See Also
processingChIP

Examples
showClass("ChIPScore")
chipSd

Accessory method for chipSd slot in a parameterOptions object.

Description

Access or Extract chipSd slot in a parameterOptions object.

Usage

chipSd(object)

Arguments

object object is a parameterOptions

Details

When computing ChIP-seq like profiles (computeChIPProfile, the occupancy values given by computeOccupancy are transformed into ChIP-seq like profiles. The average size of a ChIP-seq peak was described by Kaplan (Kaplan et al., 2011). The average peak size is subject to variation. This variation is accounted for with chipSd. It is advised to use the standard deviation of ChIP peak width from actual ChIP-seq data. We strongly encourage to set values when building a parameterOptions object.

Value

Returns a parameterOptions object with an updated value for chipSd.

Author(s)

Patrick C.N. Martin <pcnmartin@gmail.com>

References


Examples

# Building parameterOptions object
OPP <- parameterOptions()
# Accessing chipSd slot
chipSd(OPP)
Methods for function `chipSd`

## Description

Setter methods for `chipSd` slot in a `parameterOptions` object.

## Usage

```r
chipSd(object) <- value
```

## Arguments

- `object` : object is `parameterOptions` object.
- `value` : value is a positive numeric value that will be assigned to `chipSd` slot. Default value is 150.

## Details

When computing ChIP-seq like profiles (see `computeChIPProfile`), the occupancy values given by `compute0occupancy` are transformed into ChIP-seq like profiles. The average size of a ChIP-seq peak was described by Kaplan (Kaplan et al., 2011). The average peak size is subject to variation. This variation is accounted for with `chipSd`. It is advised to use the standard deviation of ChIP peak width from actual ChIP-seq data. We strongly encourage to set values when building a `parameterOptions` object.

## Value

Returns a `parameterOptions` object with an updated value for `chipSd`.

## Author(s)

Patrick C.N. Martin <pcnmartin@gmail.com>
References

Examples
# Building parameterOptions object
OPP <- parameterOptions()
# Setting new value for chipSd slot
chipSd(OPP) <- 250

Methods:

chipSd<-methods

chipSd<–methods

chipSmooth

Accessor methods for chipSmooth slot in a parameterOptions object.

Description
Access or Extract chipSmooth slot in a parameterOptions object.

Usage
chipSmooth(object)

Arguments

object object is a parameterOptions object.

Details
When computing ChIP-seq like (computeChIPProfile) profile from occupancy data (see computeOccupancy), the profiles are smoothed using a window of a given size. The default value is set at 250 base pairs. If chipSmooth is set to 0 then the profile will not be smoothed. We strongly encourage to set values when building a parameterOptions object.
Value

Returns the chipSmooth slot in an `parameterOptions` object.

Author(s)

Patrick C.N. Martin <pcnmartin@gmail.com>

References


Examples

```r
# Building parameterOptions object
OPP <- parameterOptions()
# Accessing chipSd slot
chipSmooth(OPP)
```

Description

~~ Methods for Function `chipSmooth` ~~

Methods:

signature(object = "parameterOptions")

```
chipSmooth<-  

Setter method for chipSmooth slot in parameterOptions object.
```

Usage

`chipSmooth(object) <- value`

Arguments

- `object`:
  - object is a `parameterOptions` object.
- `value`:
  - value is the positive numeric value to be assigned to the chipSmooth slot in `parameterOptions` Default value is 250 base pairs.
Details

When computing ChIP-seq like (computeChIPProfile) profile from occupancy data (see computeOccupancy), the profiles are smoothed using a window of a given size. The default value is set at 250 base pairs. If chipSmooth is set to 0 then the profile will not be smoothed. We strongly encourage to set values when building a parameterOptions object.

Value

Returns a parameterOptions object with an updated value for chipSmooth slot.

Author(s)

Patrick C.N Martin <pm16057@essex.ac.uk>

References


Examples

# Building parameterOptions object
OPP <- parameterOptions()
# Setting new value for chipSd slot
chipSmooth(OPP) <- 250
computeChIPProfile is a function that computes ChIP-seq like profiles from occupancy data. It is computed using the `computeOccupancy` function.

### Usage

```r
computeChIPProfile(genomicProfiles, loci, parameterOptions = NULL,
                    norm = TRUE, method = c("moving_kernel","truncated_kernel","exact"),
                    peakSignificantThreshold= NULL,cores=1, verbose = TRUE)
```

### Arguments

- **genomicProfiles**
  - is the result of `computeOccupancy`. This object should be a genomicProfiles object.

- **loci**
  - is either a GRanges or ChIPScore object. ChIPScore-class will be the result of `processingChIP`. This object represents the set of Loci you are interested in analysing. If you have followed the full ChIPanalyser pipe line, you would have used the processingChIP function that would return a ChIPScore-class object containing your loci of interest. GRanges are also supported if you are only using part of the pipeline.

- **parameterOptions**
  - is a parameterOptions object. This object is used to store the numerous paramters offered by ChIPanalyser. This argument is optional as all arguments are also parse in both ChIPScore-class and genomicProfiles objects. If you wanted to make some last minute changes, parameterOptions is the way to go. We recommand that you set your desired options before hand.

- **norm**
  - is a logical value. If TRUE, the ChIP-seq like profile will be normalised towards maximum Occupancy. If FALSE, the profile will be left as is.

- **method**
  - is a character string of one of the following: c("moving_kernel","truncated_kernel","exact"). If set to moving_kernel, the peaks will be approximated using Rcpp (Default). If set to truncated_kernel, the peaks will be approximated however this method does not require Rcpp. If set to exact, the peaks will not be approximated.

- **peakSignificantThreshold**
  - is a threshold at which peaks will be selected. IMPORTANT: if you select "moving_kernel" as described in method then this threshold is a numeric value describing the peak tail hight cutoff value (Default = 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 (Default = 1250). The default is set to NULL in this function. This just means that either the value is provided bu user with the appropriate method. If not, the default will be selected depending on the method selected.
computeChIPProfile

cores cores is the number of cores that will be used to compute ChIP profiles.
verbose verbose is a logical value. If TRUE, progress messages will be displayed in console. If FALSE, no progress messages will be displayed in console.

Details

computeChIPProfile converts Transcription Factor occupancy to a profile resembling the one of a ChIP-seq profile. Internally a few parameters are required to build a ChIP-like profile. These parameters are either defined and stored in a ChIPScore object (Parameters are updated based on your ChIP data), a genomicProfiles (user defined at the start of the analysis) or a parameterOptions (if you want to update values as you go along)

Value

Returns a genomicProfiles object containing all ChIP-seq like profile for every combination of lambdaPWM and boundMolecules provided by the user.

Author(s)

Patrick C.N. Martin <pcnmartin@gmail.com>

References


Examples

#Extracting Data
data(ChIPanalyserData)
# path to Position Frequency Matrix
PFM <- file.path(system.file("extdata",package="ChIPanalyser"),"BEAF-32.pfm")
#As an example of genome, this example will run on the Drosophila genome

if(!require("BSgenome.Dmelanogaster.UCSC.dm6", character.only = TRUE)){
  if (!requireNamespace("BiocManager", quietly=TRUE))
    install.packages("BiocManager")
  BiocManager::install("BSgenome.Dmelanogaster.UCSC.dm6")
}
library(BSgenome.Dmelanogaster.UCSC.dm6)
DNASequenceSet <- getSeq(BSgenome.Dmelanogaster.UCSC.dm6)
# Building genomicProfiles object
GPP <- genomicProfiles(PFM=PFM, PFMFormat="JASPAR", BPFrequency=DNASequenceSet)

# Computing Genome Wide
GenomeWide <- computeGenomeWideScores(genomicProfiles = GPP, 
                                        DNASequenceSet = DNASequenceSet)
#Compute PWM Scores
PWMScores <- computePWMScore(genomicProfiles = GenomeWide, 
                                DNASequenceSet = DNASequenceSet, loci = top, chromatinState = Access)

#Compute Occupancy
Occupancy <- computeOccupancy(genomicProfiles = PWMScores)

#Compute ChIP profiles
chipProfile <- computeChIPProfile(genomicProfiles=Occupancy, loci=top)
chipProfile

computeGenomeWideScores

\textit{Computing Genome Wide scores}

\section*{Description}
computeGenomeWideScores compute the max and min PWM score over the entire genome.

\section*{Usage}
computeGenomeWideScores(genomicProfiles, DNASequenceSet, chromatinState = NULL, parameterOptions = NULL, cores = 1, verbose = TRUE)

\section*{Arguments}
\begin{enumerate}
    \item \textbf{genomicProfiles} \\
        genomicProfiles is a genomicProfiles object containing the PFM, PWM of interest.
    \item \textbf{DNASequenceSet} \\
        DNASequenceSet is a BSgenome or DNAStringSet containing the sequence of the organism of interest.
    \item \textbf{chromatinState} \\
        chromatinState is a GRanges object containing the chromatin States. This can either represent regions of accessible DNA or Chromatin state affinities.
    \item \textbf{parameterOptions} \\
        parameterOptions is a parameterOptions object containing parameters that you wish to change. The genomicProfiles object will be updated using the values assigned to parameterOptions
    \item \textbf{cores} \\
        cores is the number or cores that will be used (Numeric value - Default = 1 )
    \item \textbf{verbose} \\
        verbose is a logical value that will determine if internal progress message will be printed.
\end{enumerate}

\section*{Details}
computeGenomeWideScores function computes PWM scores over the entire genome (or accessible Genome if chromatin State are provided ). Genome wide scores are used to determine the maximum and minimum PWM score as well as the average exponential score. These scores will in turn be used to determine which score are above the PWM thershold. The average exponential score is an integrale part of the equation used to compute Occupancy. Using default settings, ChIPanalyser will only compute occupancy on the top 70% of PWM scores. This threshold can be changed. See \textit{PWMThreshold}
computeOccupancy

Value

Returns a genomicsProfiles object with updated values for max score, min score and average-ExpPWMScore.

Author(s)

Patrick C.N Martin <pm16057@essex.ac.uk>

References


Examples

```r
if(!require("BSgenome.Dmelanogaster.UCSC.dm6", character.only = TRUE)){
  if (!requireNamespace("BiocManager", quietly=TRUE))
    install.packages("BiocManager")
  BiocManager::install("BSgenome.Dmelanogaster.UCSC.dm6")
}
library(BSgenome.Dmelanogaster.UCSC.dm6)
DNASequenceSet <- getSeq(BSgenome.Dmelanogaster.UCSC.dm6)
# Building genomicProfiles object
GPP <- genomicProfiles(PFM=PFM,PFMFormat="JASPAR", BPFrequency=DNASequenceSet)
# Computing Genome Wide
GenomeWide <- computeGenomeWideScores(genomicProfiles = GPP,
                                           DNASequenceSet = DNASequenceSet)
```

computeOccupancy

Compute Occupancy values from PWM Scores based on model.

Description

computeOccupancy will compute the Occupancy from PWM Scores. As described in detail in the vignette, ChIPanalyser uses PWM Scores, DNA Accessibility data, the number of bound molecules and a scaling factor of Transcription Factor specificity. This function will compute occupancy using the values assigned to each variable.

Usage

```r
computeOccupancy(genomicProfiles,parameterOptions = NULL,
                  norm = TRUE, verbose = TRUE)
```
computeOccupancy

**Arguments**

- **genomicProfiles**
  
  genomicProfiles is a genomicProfiles object resulting from `computePWMScore`. IT is important to use this resulting object as the occupancy will only be computed for sites above a threshold.

- **parameterOptions**
  
  parameterOptions is a parameterOptions object containing the adequate values assigned to each Parameter. If not supplied (parameterOptions = NULL), a new object will be created internally using default values.

- **norm**
  
  norm a logical value which determines if the occupancy should be normalised or not.

- **verbose**
  
  verbose a logical value which determines if progress messages are printed or not.

**Details**

computeOccupancy will compute the Occupancy from PWM Scores. As described in detail in the vignette, ChIPanalyser uses PWM Scores, DNA Accessibility data, the number of bound molecules and a scaling factor of Transcription Factor specificity. This function will compute occupancy using the values assigned to each variable. It should also be noted that the parameterOptions object contains a set of parameters used to compute Occupancy (not only restricted to this). These parameters are often dependant on real ChIP-Seq data and will influence the goodness of fit between the predicted model and real ChIP-seq data. We strongly advise that the values assigned to each parameter should be customised in order to increase the model agreement with real world biological data.

**Value**

computeOccupancy will return a genomicProfiles. The main difference will reside in the profiles slot. This slot is generally a list or GRangesList. Within these list type structures are enclosed GRanges containing the positions of site above threshold, PWMScores and Occupancy for each site. The series of GRanges will depend on the number of loci that are tested and the number of element in the list will depend on the various combinations of lambdaPWM and boundMolecules.

**Author(s)**

Patrick C.N. Martin <pcnmartin@gmail.com>

**References**


**Examples**

```r
# Data extraction
data(ChIPanalyserData)
# path to Position Frequency Matrix
PFM <- file.path(system.file("extdata",package="ChIPanalyser"),"BEAF-32.pfm")
```
#As an example of genome, this example will run on the Drosophila genome

if(!require("BSgenome.Dmelanogaster.UCSC.dm6", character.only = TRUE)){
  if (!requireNamespace("BiocManager", quietly=TRUE))
    install.packages("BiocManager")
  BiocManager::install("BSgenome.Dmelanogaster.UCSC.dm6")
}
library(BSgenome.Dmelanogaster.UCSC.dm6)
DNASequenceSet <- getSeq(BSgenome.Dmelanogaster.UCSC.dm6)

#Building data objects
GPP <- genomicProfiles(PFM=PFM,PFMFormat="JASPAR",BPFrequency=DNASequenceSet)
OPP <- parameterOptions()
# Computing Genome Wide
GenomeWide <- computeGenomeWideScores(genomicProfiles = GPP, DNASequenceSet = DNASequenceSet)

#Compute PWM Scores
PWMScores <- computePWMScore(genomicProfiles = GenomeWide, DNASequenceSet = DNASequenceSet, loci = top, chromatinState = Access)

#Compute Occupancy
Occupancy <- computeOccupancy(genomicProfiles = PWMScores, parameterOptions = OPP)
Occupancy

---

### computeOptimal

#### compute Optimal Parameters

**Description**

ChIPanalyser contains a set of functions some of which require two parameters known as \texttt{lambdaPWM} and as \texttt{boundMolecules}. These two parameters are not always known. \texttt{computeOptimal} will compute these values by maximising the correlation and minimising the Mean Squared Error between a predicted ChIP-seq-like profile and a real ChIP-seq profile for a given loci.

**Usage**

```r
computeOptimal(genomicProfiles,DNASequenceSet, ChIPScore, chromatinState = NULL, parameterOptions = NULL, optimalMethod = "all", rank=FALSE, returnAll=TRUE, peakMethod="moving_kernel", cores=1)
```

**Arguments**

- **genomicProfiles**
  - \texttt{genomicProfiles} is a \texttt{genomicProfiles} object containing at least a Postion Frequency Matrix or a Position Weight Matrix. It is strongly advised to cus-
to optimize this object to increase goodness of fit of the model when compared to real ChIP-seq data.

DNASequenceSet     DNASequenceSet is a DNAStringSet or a BSgenome of the full sequence of the organism of interest.

ChIPScore          ChIPScore is a named list containing ChIP-seq enrichments for each Loci of interest. This Profile should be normalised to a base pair level. In other words, there should be an enrichment score for each base pair of a given Locus.

chromatinState     chromatinState is a GRanges object containing either accessible sites or DNA affinity scores.

parameterOptions   parameterOptions is a parameterOptions object. If this object is not provided (parameterOptions = NULL), a new object will be created internally. However, it is strongly advised to tailor this object to maximise the goodness of fit of the model when compared to ChIP-seq data.

optimalMethod      optimalMethod is a character string which determines which method for optimal parameter selection should be selected. optimalMethod can be one of the following: pearson, spearman, kendall, ks, fscore, geometric, MSE, or all. Default is set to all.

rank               rank is a logical value indicating if optimal parameters should be based on rank (parameter combination occurring the most over all regions) or average score (best performing combination of parameters on average over all regions selected). DEFAULT = FALSE

returnAll          returnAll is a logical value indicating if all internal objects should be returned. DEFAULT = TRUE. Internal objects are the following: Occupancy Scores, ChIP like profiles, goodness of fit metrics and optimal parameters. If set to FALSE, computeOptimal will only return the optimal parameters.

peakMethod         peakMethod is a character string of one of the following: c("moving_kernel", "truncated_kernel", "exact"). If set to moving_kernel, the peaks will be approximated using Rcpp (Default). If set to truncated_kernel, the peaks will be approximated however this method does not require Rcpp. If set to exact, the peaks will not be approximated.

cores              cores is the number cores that will be used to compute optimal set of parameters.

Details

In order to backward infer the values of lambdaPWM and boundMolecules, it is possible to use the computeOptimal to find these parameters. It should be noted that this function requires a ChIP-seq data input. ChIPScore (ChIP-seq data). This should be the output of the processingChIP function.

Value

computeOptimal returns a list respectively described as the optimal set of Parameters (lambda - lambdaPWM and boundMolecules), the optimal matrix (a matrix containing accuracy estimates dependant on the parameter chosen), and finally the chosen parameter. If the parameter that was chosen was "all", then each element of this list will contain the optimal set of parameters, optimal matrices for all of the aforementioned parameters (see optimalMethod).
Author(s)
Patrick C. N. Martin <pm16057@essex.ac.uk>

References

Examples

```r
#Data extraction
data(ChIPanalyserData)

# path to Position Frequency Matrix
PFM <- file.path(system.file("extdata",package="ChIPanalyser"),"BEAF-32.pfm")

#As an example of genome, this example will run on the Drosophila genome

if(!require("BSgenome.Dmelanogaster.UCSC.dm6", character.only = TRUE)){
  if (!requireNamespace("BiocManager", quietly=TRUE))
    install.packages("BiocManager")
  BiocManager::install("BSgenome.Dmelanogaster.UCSC.dm6")
}

library(BSgenome.Dmelanogaster.UCSC.dm6)
DNASequenceSet <- getSeq(BSgenome.Dmelanogaster.UCSC.dm6)
chip <- processingChIP(chip,top)

#Building data objects
GPP <- genomicProfiles(PFM=PFM,PFMFormat="JASPAR",BPFrequency=DNASequenceSet)
OPP <- parameterOptions()

#Computing Optimal set of Parameters
optimalParam <- computeOptimal(genomicProfiles = GPP,
                               DNASequenceSet = DNASequenceSet,
                               ChIPScore = chip,
                               chromatinState = Access,
                               parameterOptions = OPP,
                               parameter = "all",
                               peakMethod="moving_kernel")
```

computePWMScore

Compute PWM Scores of sites above threshold.

Description

computePWMScore will compute and extract all sites that exhibit a PWM Score higher than a threshold. This threshold (see pwmThreshold) will determine the percentage of total sites that should NOT be considered.

Usage

```r
computePWMScore(genomicProfiles,DNASequenceSet,
                 loci = NULL, chromatinState = NULL, parameterOptions=NULL, cores=1, verbose = TRUE)
```
computePWMScore

Arguments

DNASequenceSet DNASequenceSet is a DNAStringSet or a BSgenome containing the full sequence of the organism of interest.

genomicProfiles genomicProfiles is a genomicProfiles object resulting from the computeGenomeWideScores function.

loci loci is a GRanges object containing the Loci of interest or a ChIPScore object result of processingChIP function.

parameterOptions parameterOptions is a parameterOptions object containing parameters that you wish to parse/change when computing PWMScores.

chromatinState chromatinState is a GRanges object sites of accessible DNA or DNA affinity scores.

cores cores is the number of cores used to compute PWM Scores.

verbose verbose is a logical value indicating if progress messages should be printed or not.

Details

After determining genome wide scores, it is possible to only compute and extract high affinity sites (in the sense that they have a high PWM Score). If a PWMThreshold is not set by user, the default value is set at 0.7. This means that 70% of sites will NOT be selected. Only the top 30% will be computed and extracted. If one is interested in all PWM Scores at a genome wide scale (or accessible DNA), this is possible by setting PWMThreshold to zero.

Value

computePWMScore will return a genomicProfiles object. The profiles slot will have been updated. This slot will now contain a GRangesList with each element being a GRanges. This GRanges will contain postion of each sites (start, end and strand) and the PWMScore associated to that site.

Author(s)

Patrick C.N. Martin <pcnmartin@gmail.com>

References


Examples

#Data extraction
data(ChIPanalyserData)

# path to Position Frequency Matrix
PFM <- file.path(system.file("extdata",package="ChIPanalyser"),"BEAF-32.pfm")

#As an example of genome, this example will run on the Drosophila genome
if(!require("BSgenome.Dmelanogaster.UCSC.dm6", character.only = TRUE)){
  if (!requireNamespace("BiocManager", quietly=TRUE))
    install.packages("BiocManager")
  BiocManager::install("BSgenome.Dmelanogaster.UCSC.dm6")
}
library(BSgenome.Dmelanogaster.UCSC.dm6)
DNASequenceSet <- getSeq(BSgenome.Dmelanogaster.UCSC.dm6)
chip<-processingChIP(chip,top)
#Building data objects
GPP <- genomicProfiles(PFM=PFM,PFMFormat="JASPAR",BPFrequency=DNASequenceSet)

# Computing Genome Wide
GenomeWide <- computeGenomeWideScores(DNASequenceSet = DNASequenceSet,
    genomicProfiles = GPP)

#Compute PWM Scores
PWMScores <- computePWMScore(DNASequenceSet = DNASequenceSet,
    genomicProfiles = GenomeWide,
    loci = chip, chromatinState = Access)
PWMScores

---

**DNASequenceLength**  
*Accessor method for DNASequenceLength slot in a genomicProfiles*

**Description**

Accessor method for DNASequenceLength slot in a genomicProfiles

**Usage**

DNASequenceLength(object)

**Arguments**

object object is a genomicProfiles

**Details**

The model on which is based ChIPanalyser requires the length of the DNA sequence used to compute scores. In this circumstance, this DNA Length is the total length of the DNA of the organism of interest or the the Accessible DNA at a genome wide scale.

**Value**

Returns DNASequenceLength slot in a genomicProfiles object.
Author(s)

Patrick C. N. Martin <p.martin@essex.ac.uk>

References


Examples

# Data extraction
data(ChIPanalyserData)
# path to Position Frequency Matrix
PFM <- file.path(system.file("extdata",package="ChIPanalyser"),"BEAF-32.pfm")
# As an example of genome, this example will run on the Drosophila genome

if(!require("BSgenome.Dmelanogaster.UCSC.dm6", character.only = TRUE)){
  if (!requireNamespace("BiocManager", quietly=TRUE))
    install.packages("BiocManager")
  BiocManager::install("BSgenome.Dmelanogaster.UCSC.dm6")
}
library(BSgenome.Dmelanogaster.UCSC.dm6)
DNASequenceSet <- getSeq(BSgenome.Dmelanogaster.UCSC.dm6)

# Building genomicProfiles object
GPP <- genomicProfiles(PFM=PFM,PFMFormat="JASPAR",BPFrequency=DNASequenceSet)
# Computing Genome Wide
GenomeWide <- computeGenomeWideScores(DNASequenceSet = DNASequenceSet, genomicProfiles = GPP)

DNASequenceLength(GenomeWide)

---

DNASequenceLength-methods

~~ Methods for Function DNASequenceLength ~~

Description

~~ Methods for function DNASequenceLength ~~

Methods:

signature(object = "genomicProfilesInternal")
**Description**

Accessor Method for the `drop` slot in a `genomicProfiles` object.

**Usage**

`drop(object)`

**Arguments**

- `object` object is a `genomicProfiles` object.

**Details**

During certain computations, it is possible that the Loci of interest do no show any overlap with accessible DNA. If this were to be the case, a warning message will appear in the console but these inaccessible Loci will be stored in this slot. It is also for these reasons that it is imperative for Loci of interest to be named (in this case, a named `GRanges`).

**Value**

Returns a character string with loci containing no accessible DNA.

**Author(s)**

Patrick C.N. Martin <p.martin@essex.ac.uk>

**References**


**Examples**

```r
data(ChIPanalyserData)
#Loading PFM files
PFM <- file.path(system.file("extdata",package="ChIPanalyser"),"BEAF-32.pfm")
#Building data objects
GPP <- genomicProfiles(PFM=PFM,PFMFormat="JASPAR")

# Loci with no acces - a warning message will be issued
# if loci do no contain accesible DNA
# Otherwise this slot will remain empty
drop(GPP)
```
Methods:

signature(object = "genomicProfilesInternal")

Description

Running the ChIPanalyser implementation of a Genetic algorithm.

evolve pushes a starting population to evolve in a genetic algorithm.

Usage

evolve(population,DNASequenceSet,ChIPScore, genomicProfiles,parameters=NULL,generations=100,mutationProbability=0.3, offsprings=5,chromatinState=NULL, method="geometric", lambda=TRUE, checkpoint=TRUE, filename=NULL, cores=1)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>population</td>
<td>numeric value describing the number of individuals in the starting population. Alternatively - a starting population list as returned by generateStartingPopulation. NOTE: if numeric - the parameter argument is also required.</td>
</tr>
<tr>
<td>DNASequenceSet</td>
<td>DNAStringSet object containing DNA sequences of interest (Extracted from BSGenome)</td>
</tr>
<tr>
<td>ChIPScore</td>
<td>ChIPScore object as returned by the processingChIPfunction</td>
</tr>
<tr>
<td>genomicProfiles</td>
<td>genomicProfiles object containing minimal information (such as the PWM)</td>
</tr>
<tr>
<td>parameters</td>
<td>vector or list containing each parameter that should be added to the chromosome. See generateStartingPopulation</td>
</tr>
<tr>
<td>generations</td>
<td>numeric describing the number of generation before the Genetic algorithm should halt.</td>
</tr>
<tr>
<td>mutationProbability</td>
<td>numeric describing the rate of mutations for each surviving individual</td>
</tr>
<tr>
<td>offsprings</td>
<td>numeric describing the number of individuals surviving to the next generation</td>
</tr>
</tbody>
</table>
chromatinState  GRanges object containing chromatin state information. Each state should be labeled in a meta data column named "name". It is advised to use numeric values for each state name.

method  character string describing the scoring metric that should be used. ChIPanalyzer offers twelve different metrics: correlation coefficients (Pearson, Spearman and Kendall), Mean Squared Error (MSE), Kolmogorov–Smirnov Distance, precision, recall, accuracy, F-score, Matthew’s correlation coefficient (MCC) and Area Under Curve Receiver Operator Characteristic (AUC ROC or just AUC)

lambda  logical describing if lambda value should be pre-computed. Setting to TRUE increases the speed of the algorithm.

checkpoint  logical describing if population parameters at each generations should be saved.

filename  character string that will serve as a prefix to the saved intermediate files.

cores  numeric describing the number of cores used to run the GA.

Details

ChIPanalyser offers a way of finding optimal solution by using a genetic algorithm. Instead of running the standard analysis, TF binding affinities to chromatin states can be extracted via this more complex method. It should be noted that this method is better suited for the analysis of chromatin states. While the algorithm still works with simple DNA Accessibility, it would potentially take more time for accuracy minor gains.

Value

Returns a named list with three elements.

- database saves the data frame containing all scores for each individual since generation 1
- population saves the last population with chromosome values
- fitest saves the fittest individual for a given generation

Author(s)

Patrick C.N. Martin <pcnmartin@gmail.com

Examples

library(ChIPanalyser)
data(ChIPanalyserData)
# See GA vignette for usage
generateStartingPopulation

Generate Starting population for ChIPanalyser Genetic algorithm

Description

generateStartingPopulation generates a starting population with random traits for each individual.

Usage

generateStartingPopulation(population, parameters, names=NULL)

Arguments

population numeric value describing the number of individuals in the starting population.
parameters vector or list containing each parameter that should be added to the chromosome.
names character describing names that should be added to each individual.

Details

generateStartingPopulation generates a starting population to be used in the genetic algorithm implemented in ChIPanalyser. There are two main ways a starting population can be generated:

1. by name Using names of each parameter that should be parse to each "chromosome". The possible parameters are N, lambda, PWMThreshold, CS (DNAAffinity or DNAAccessibilty also works). CS values should also contain a numeric value associated to each chromatin state you wish to parse. e.g CS1 ... CS14 This will generate a value by sampling from a set of predefined value for each parameters.

2. by value range Using a named list (names for each parameters). Each element of the list should contain three numeric values : length of range, min value, max value. (Internally - values are parse to runif)

Value

Returns a list of individuals with a random traits

Author(s)

Patrick C.N. Martin
### genomicProfiles

#### Description

**genomicProfiles** is an S4 object serving two purposes: (i) storing internal computed data and (ii) storing parameter options. This object is parsed through the different steps of the pipeline to facilitate that parsing and changing of parameters.

#### Usage

```r
genomicProfiles(..., parameterOptions = NULL, genomicProfiles = NULL, ChIPScore = NULL)
```

#### Arguments

- `...` Any of the user available slots in genomicProfiles.
- `parameterOptions` If some parameters were already previously computed or stored in a parameterOptions, parsing this object will use those values instead of the default ones.
- `genomicProfiles` If some parameters were already previously computed or stored in a genomicProfiles, parsing this object will use those values instead of the default ones.
- `ChIPScore` If some parameters were already previously computed or stored in a ChIPScore, parsing this object will use those values instead of the default ones.

#### Details

The genomicProfiles object serves the purpose of storing, and parsing parameters and computed data between the different steps of the pipeline. When creating a genomicProfiles object it is possible to use previously computed values by simply parsing the object to the constructor function.

#### Value

Returns a genomicProfiles object with updated slots for all parameters parsed.
Author(s)
Patrick C. N. Martin <pm16057@essex.ac.uk>

References

See Also
genomicProfiles
parameterOptions

Examples

```r
PFM <- file.path(system.file("extdata",package="ChIPanalyser"),"BEAF-32.pfm")
genomicProfiles()
genomicProfiles(PFM=PFM,PFMFormat="JASPAR")
```

description

GenomicProfiles is an S4 object serving two purposes: (i) storing internal computed data and (ii) storing parameter options. This object is parsed through the different steps of the pipeline to facilitate that parsing and changing of parameters.

Objects from the Class

Objects can be created by calls of the form genomicProfiles(ploidy, boundMolecules, backgroundSignal, maxSignal, lociWidth, chipMean, chipSd, chipSmooth, stepSize, noiseFilter, removeBackground, lambdaPWM, PWMpseudocount, naturalLog, noOfSites, PWMThreshold, strandRule, whichstrand, PFM, PWM, PFMFormat, BPFrequency, minPWMscore, maxPWMscore, profiles, DNASequenceLength, averageExpPWMscore).

Slots

- **PWM**: Object of class "matrix": A Position Weight Matrix (either supplied or internally computed if PFM is provided)
- **PFM**: Object of class "matrix": A Position Frequency Matrix (may also be a path to file containing PFM)
- **PFMFormat**: Object of class "character": A character string of one of the following: raw, transfac,JASPAR or sequences
BPFrequency: Object of class "vector": Base Pair Frequency in the genome (if a DNA sequence is provided (as a DNAStringSet or BSgenome), will be automatically computed internally). Default: c(0.25,0.25,0.25,0.25)

minPWMScore: Object of class "vector": Lowest PWM score across the genome (computed and updated internally)

maxPWMScore: Object of class "vector": Highest PWM score across the genome (computed and updated internally)

profiles: Object of class "GRList": Contains GRanges with sites above threshold and associated metrics (PWMscore and Occupancy) - Computed Internally

DNASequenceLength: Object of class "vector": Length of the Genome (or accessible genome) - computed internally

averageExpPWMScore: Object of class "vector": Average exponential PWM score across the genome (or accessible genome) - computed internally

ZeroBackground: Object of class "vector": Internal background value (computed internally)

drop: Object of class "vector": Stores Loci that do contain accessible DNA if it were to be the case (computed and updated internally)

tags: Object of class "character" - Internal Tags-

ploidy: Object of class "numeric": A numeric Value describing the ploidy of the organism. Default: 2

boundMolecules: Object of class "vector": A vector (or single value) containing the number of bound Molecules (bound Transcription Factors). Default: 1000

backgroundSignal: Object of class "numeric": A numeric value describing the ChIP-seq background Signal (average signal from real ChIP seq data). Default: 0

maxSignal: Object of class "numeric": A numeric value describing the highest ChIP-seq signal (from real ChIP-seq data). Default: 1

lociWidth: Object of class "numeric" ~

chipMean: Object of class "numeric": A numeric value describing the mean width of a ChIP-seq peak. Default: 150

chipSd: Object of class "numeric": A numeric value describing the standard deviation of ChIP-seq peaks. Default: 150

chipSmooth: Object of class "vector": A numeric value describing the width of the window used to smooth Occupancy profiles into ChIP profiles. Default: 250

stepSize: Object of class "numeric": A numeric value describing the step Size (in base pairs) between each ChIP-seq score. Default: 10 (Scored every 10 base pairs)

removeBackground: Object of class "numeric": A numeric value describing the value at which score should be removed. Default: 0 (If negative scores then remove)

noiseFilter: Object of class "character" ~ Describes the noiseFilter method that will be applied to ChIP data (Zero, mean, median, sigmoid)~

PWMThreshold: Object of class "numeric": Threshold at which PWM Score should be selected (only sites above threshold will be selected - between 0 and 1)

strandRule: Object of class "character": "mean", "max" or "sum" will determine how strand should be handle for computing PWM Scores. Default: "max"
whichstrand: Object of class "character": "+", ",-" or ",+-" on which strand should PWM Score be computed. Default: ",+-"

lambdaPWM: Object of class "vector" A vector (or single value) containing values for lambdaPWM
Default: 1

naturalLog: Object of class "logical": A logical value describing if natural Log will be used to compute the PWM (if FALSE then log2 will be used). Default: TRUE

noOfSites: Object of class "nos" A Positive integer describing number of sites (in base pair) should be used from the PFM to compute PWM. Default = 0 (Full width of binding site will be used when set to 0)

PWMpseudocount: Object of class "numeric": A numeric value describing a PWMpseudocount for PWM computation. Default: 1

paramTag: Object of class "character" ~Internal~

Extends
Class "genomicProfilesInternal", directly. Class "parameterOptions", directly.

Methods
initialize signature(.Object = "genomicProfiles"): ...
show signature(object = "genomicProfiles"): ...

Author(s)
Patrick C. N. Martin <p.martin@essex.ac.uk>

References

See Also

genomicProfiles parameterOptions

Examples
showClass("genomicProfiles")
genomicProfilesInternal-class

Class "genomicProfilesInternal"

Description

Non exported class. Represents the stripped down version of genomicProfiles.

Objects from the Class

Created Internally.

Slots

PWM: Object of class "matrix" ~
PFM: Object of class "matrix" ~
PFMFormat: Object of class "character" ~
BPFrequency: Object of class "vector" ~
minPWMScore: Object of class "vector" ~
maxPWMScore: Object of class "vector" ~
profiles: Object of class "GRList" ~
DNASequenceLength: Object of class "vector" ~
averageExpPWMScore: Object of class "vector" ~
ZeroBackground: Object of class "vector" ~
drop: Object of class "vector" ~
tags: Object of class "character" ~

Methods

.averageExpPWMScore<- signature(object = "genomicProfilesInternal", value = "numeric"): ...
.DNASequenceLength<- signature(object = "genomicProfilesInternal", value = "vector"): ...
.drop<- signature(object = "genomicProfilesInternal", value = "vector"): ...
.generatePWM signature(object = "genomicProfilesInternal"): ...
.maxPWMScore<- signature(object = "genomicProfilesInternal", value = "vector"): ...
.minPWMScore<- signature(object = "genomicProfilesInternal", value = "vector"): ...
.profiles<- signature(object = "genomicProfilesInternal", value = "GRList"): ...
.tags signature(object = "genomicProfilesInternal"): ...
.tags<- signature(object = "genomicProfilesInternal", value = "character"): ...
averageExpPWMScore signature(object = "genomicProfilesInternal"): ...
genomicProfilesInternal-class

BPFrequency signature(object = "genomicProfilesInternal"): ...
BPFrequency<- signature(object = "genomicProfilesInternal", value = "DNAStringSet"): ...
BPFrequency<- signature(object = "genomicProfilesInternal", value = "vector"): ...
DNASequenceLength signature(object = "genomicProfilesInternal"): ...
drop signature(object = "genomicProfilesInternal"): ...
maxPWMscore signature(object = "genomicProfilesInternal"): ...
minPWMscore signature(object = "genomicProfilesInternal"): ...
PFMFormat signature(object = "genomicProfilesInternal"): ...
PFMFormat<- signature(object = "genomicProfilesInternal", value = "character"): ...
PositionFrequencyMatrix signature(object = "genomicProfilesInternal"): ...
PositionFrequencyMatrix<- signature(object = "genomicProfilesInternal", value = "character"): ...
PositionFrequencyMatrix<- signature(object = "genomicProfilesInternal", value = "matrix"): ...
PositionWeightMatrix signature(object = "genomicProfilesInternal"): ...
PositionWeightMatrix<- signature(object = "genomicProfilesInternal", value = "matrix"): ...
profiles signature(object = "genomicProfilesInternal"): ...

Author(s)

Patrick C. N. Martin <pm16057@essex.ac.uk>

References


See Also

genomicProfiles
parameterOptions

Examples

showClass("genomicProfilesInternal")
getHighestFitnessSolutions

Get Highest Fitness Solutions

Description

generateFitnessSolutions extract best solution from a ChIPanalyser GA/evolve Run.

Usage

generateFitnessSolutions(population, child=2, method="geometric")

Arguments

- population: Population list as output by the evolve function.
- child: numeric describing the number of solution to be extracted from Population list.
- method: character string describing which scoring method should be used and selected from "geometric", "ks", "MSE", " Pearson", " spearman", " kendall", " recall", " precision", " fscore", " MCC", " Accuracy" or "AUC".

Details

This function only serves as a way of extracting data from the population list. Ultimately - it is just a wrapper for some indexing.

Value

Return the index of the top "child" solutions.

Author(s)

Patrick C.N. Martin <pcnmartin@gmail.com>

Examples

library(ChIPanalyser)
data(ChIPanalyserData)
# See GA vignette for usage
**getTestingData**

*Extract testing data from ChIPscore object*

**Description**

getTestingData extracts selected regions from ChIPscore object to be used as testing set.

**Usage**

getTestingData(ChIPscore, loci = 1)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChIPscore</td>
<td>ChIPscore object as returned by processingChIP</td>
</tr>
<tr>
<td>loci</td>
<td>numeric describing index of loci to be used as testing data.</td>
</tr>
</tbody>
</table>

**Value**

Returns ChIPscore object with the selected testing loci.

**Author(s)**

Patrick C.N. Martin <pcnmartin@gmail.com

**Examples**

```r
library(ChIPanalyser)
data(ChIPanalyserData)
# See GA vignette for usage
test <- processingChIP(chip, top)
test <- getTestingData(test, 1:2)
```

---

**getTrainingData**

*Extract training data from ChIPscore object*

**Description**

getTrainingData extracts selected regions from ChIPScore object to be used as training set.

**Usage**

getTrainingData(ChIPscore, loci = 1)
### Arguments

- **ChIPscore**: ChIPscore object as returned by `processingChIP`
- **loci**: numeric describing index of loci to be used as training data.

### Value

Returns ChIPscore object with the selected training loci.

### Author(s)

Patrick C.N. Martin <pcnmartin@gmail.com>

### Examples

```r
library(ChIPanalyser)
data(ChIPanalyserData)
# See GA vignette for usage
test <- processingChIP(chip, top)
test <- getTrainingData(test, 1:2)
```

---

### GRList-class

**Class**: "GRList"

### Description

Virtual Class to handle multiple data types for one slot (profiles)

### Objects from the Class

A virtual Class: No objects may be created from it.

### Methods

**GRList-class** The purpose of this virtual classe is to store data of two different formats in one slot: GRangesList and Lists

### Author(s)

Patrick C. N. Martin <p.martin@essex.ac.uk>

### References


### Examples

```r
showClass("GRList")
```
Description

Methods:

signature(.Object = "ChIPScore") Initialize ChIPScore
signature(.Object = "genomicProfiles") Initialize genomicProfiles
signature(.Object = "parameterOptions") Initialize parameterOptions

lambdaPWM

Usage

lambdaPWM(object)

Arguments

object object is parameterOptions object

Details

The model underlying ChIPanalyser internally infers two parameters: number of bound molecules and lambda. Lambda represents a scaling factor for the Position weight matrix (PWM). This can be described as how well does a TF discriminate between high affinity and very high affinity sites.

Value

Returns the value assigned to the lambdaPWM slot in a parameterOptions object.

Author(s)

Patrick C. N. Martin <p.martin@essex.ac.uk>

References

Examples

# Loading data
data(ChIPanalyserData)

# Building data objects
GPP <- parameterOptions(lambdaPWM=1)
# Setting new Value for lambdaPWM
lambdaPWM(GPP)

lambdaPWM-methods

Description

~~ Methods for function lambdaPWM ~~

Methods:

lambdaPWM(object)

lambdaPWM<- ~~~ Setter Method for the lambdaPWM slot in a parameterOptions object

Description

Setter Method for the lambdaPWM slot in a parameterOptions object

Usage

lambdaPWM(object)<-value

Arguments

object object is parameterOptions object
value value is the numeric value to be assigned to the lambdaPWM slot. Default set at 1.

Details

The model underlying ChIPanalyser internally infers two parameters: number of bound molecules and lambda. Lambda represents a scaling factor for the Position weight matrix (PWM). This can be described as how well does a TF discriminate between high affinity and very high affinity sites.
Value

Returns the value assigned to the lambdaPWM slot in a parameterOptions object.

Author(s)

Patrick C. N. Martin <p.martin@essex.ac.uk>

References


Examples

# Loading data
data(ChIPanalyserData)

# Building data objects
GPP <- parameterOptions(lambdaPWM=1)
# Setting new Value for lambdaPWM
lambdaPWM(GPP) <- 2

lambdaPWM<-methods

Description

Setter method for the lambdaPWM slot in the parameterOptions

Methods:

lambdaPWM(object)<-value

loci

Description

Setter Method for the loci slot in a ChIPScore object

Usage

loci(object)
Arguments

object object is ChIPScore object

Details

When using the processingChIP, this function will return a name GRanges with the loci of interest. These loci will either result from user input or extracted from the ChIP profiles (see processingChIP and lociWidth). This function enables you to extract those loci from the ChIPScore object.

Value

Returns the value assigned to the loci slot in a ChIPScore object.

Author(s)

Patrick C. N. Martin <p.martin@essex.ac.uk>

References


Examples

# Loading data
data(ChIPanalyserData)

chip<-processingChIP(chip,top)
loci(chip)
Author(s)
Patrick C. N. Martin <pm16057@essex.ac.uk>

References

See Also
ChIPScore

Examples
showClass("loci")

loci-methods

Description
Accessor method for the loci slot in ChIPScore

Methods:
loci(Object) Loci of interest parsed to or extracted from the ChIPScore object

lociWidth

Description
Accessor Method for the lociWidth slot in a parameterOptions object

Usage
lociWidth(object)

Arguments
object object is parameterOptions object
Details

When using the `processingChIP` function, the provided ChIP scores will be split into bins of a given size. `lociWidth` determines the size of that bin. Default is set at 20,000 bp. This means that the ChIP profiles provided will be split into bins of 20,000 bp over the entire profile provided if no loci of interest is provided.

Value

Returns the value assigned to the `lociWidth` slot in a `parameterOptions` object.

Author(s)

Patrick C. N. Martin <p.martin@essex.ac.uk>

References


Examples

```r
# Loading data
data(ChIPanalyserData)

# Building data objects
GPP <- parameterOptions(lociWidth=20000)
# Accessing new value for lociWidth
lociWidth(GPP)
```

Description

Accessor method for the `loci` slot in `ChIPScore`

Methods:

- `lociWidth(object)` Setting width of regions when using the `reduce` argument and NOT providing your own loci when using the `processingChIP` function.
lociWidth<-  

Setter Method for the lociWidth slot in a parameterOptions object

Description

Setter Method for the lociWidth slot in a parameterOptions object

Usage

lociWidth(object)<-value

Arguments

object  object is parameterOptions object
value   value is the numeric value to be assigned to the lociWidth slot. Default set at 1.

Details

When using the processingChIP function, the provided ChIP scores will be split into bins of a given size. lociWidth determines the Size of that bin. Default is set at 20 000 bp. This mean that the ChIP profiles provided will be split into bins of 20 000 bp over the entire profile provided if no loci of interest is provided.

Value

Returns the value assigned to the lociWidth slot in a parameterOptions object.

Author(s)

Patrick C. N. Martin <p.martin@essex.ac.uk>

References


Examples

# Loading data
data(ChIPanalyserData)

#Building data objects
GPP <- parameterOptions(lociWidth=20000)
#Setting new Value for lociWidth
lociWidth(GPP) <- 30000
lociWidth<-methods  ~~ Methods for Function lociWidth<- ~~

Description

Setter method for the loci slot in ChIPScore

Methods:

lociWidth(Object)<-value

maxPWMScore  Accessor function for maxPWMScore slot in a genomicProfiles object.

Description

Accessor function for maxPWMScore slot in a genomicProfiles object.

Usage

maxPWMScore(object)

Arguments

object object is a genomicProfiles object.

Details

maxPWMScore is a numerical value that can be described as the highest PWM score computed at a genome wide scale. This value is computed and updated in the genomicProfiles object after using the computeGenomeWideScores.

Value

Returns the value of assigned to the maxPWMScore slot in a genomicProfiles object.

Author(s)

Patrick C. N. Martin <p.martin@essex.ac.uk>

References

maxPWMScore-methods

Examples

# Loading data
# Data extraction
data(ChIPanalyserData)

# path to Position Frequency Matrix
PFM <- file.path(system.file("extdata", package="ChIPanalyser"), "BEAF-32.pfm")

# As an example of genome, this example will run on the Drosophila genome

if(!require("BSgenome.Dmelanogaster.UCSC.dm6", character.only = TRUE)){
  if (!requireNamespace("BiocManager", quietly=TRUE))
    install.packages("BiocManager")
  BiocManager::install("BSgenome.Dmelanogaster.UCSC.dm6")
}

library(BSgenome.Dmelanogaster.UCSC.dm6)
DNASequenceSet <- getSeq(BSgenome.Dmelanogaster.UCSC.dm6)

# Building data objects
GPP <- genomicProfiles(PFM=PFM, PFMFormat="JASPAR")

# Computing Genome Wide
GenomeWide <- computeGenomeWideScores(DNASequenceSet = DNASequenceSet,
genomicProfiles = GPP)
maxPWMScore(GenomeWide)

## If used before computeGenomeWidePWMScore, will return NULL

maxPWMScore-methods ~~~ Methods for Function maxPWMScore ~~~

Description

Accessor method for maxPWMScore

Methods:

maxPWMScore(object)

maxSignal ~~~ Accessor method for the maxSignal slot in a parameterOptions object. ~~~

Description

Accessor method for the maxSignal slot in a parameterOptions object.

Usage

maxSignal(object)
Arguments

object object is a parameterOptions object.

Details

In the context of ChIPanalyser, maxSignal represents the maximum normalised ChIP-Seq signal of a given Transcription factor (or DNA binding protein). Although, A default value of 1 has been assigned to this slot, we strongly recommend to tailor this value accordingly. We strongly encourage to set values when building a parameterOptions object.

Value

Returns the value assigned to the maxSignal slot in a parameterOptions object.

Author(s)

Patrick C.N. Martin <p.martin@essex.ac.uk>

References


Examples

# Building parameterOptions object
OPP <- parameterOptions()
#Setting new Value for maxSignal
maxSignal(OPP)

---

maxSignal-methods ~~~ Methods for Function maxSignal ~~~

Description

Accessor method for maxSignal

Methods:

maxSignal(object) Maximum ChIP signal extracted from ChIP data (see processingChIP)
Description

Setter method for `maxSignal` slot in a `parameterOptions` object.

Usage

```
maxSignal(object) <- value
```

Arguments

- `object` object is a `parameterOptions` object.
- `value` value is a numerical value to be assigned to the `maxSignal` slot.

Details

In the context of ChIPanalyser, `maxSignal` represents the maximum normalised ChIP-Seq signal of a given Transcription factor (or DNA binding protein). Although, a default value of 1 has been assigned to this slot, we strongly recommend to tailor this value accordingly. We strongly encourage to set values when building a `parameterOptions` object.

Value

Returns a `parameterOptions` with an updated value for `maxSignal`.

Author(s)

Patrick C.N. Martin <p.martin@essex.ac.uk>

References


Examples

```
# Building parameterOptions object
OPP <- parameterOptions()
# Setting new Value for maxSignal
maxSignal(OPP) <- 1.8
```
maxSignal<-methods  

Description

Setter method for maxSignal

Methods:

maxSignal(Object)<-value  Maximum ChIP signal extracted from ChIP data (see processingChIP)

minPWMScore  

Accessor method the minPWMScore slot in a genomicProfiles object

Description

Accessor method the minPWMScore slot in a genomicProfiles object

Usage

minPWMScore(object)

Arguments

object  object is a genomicProfiles object.

Details

minPWMScore can be described as the lowest PWM score computed at a genome wide scale. Although it is possible to assigne a value to minPWMScore, we strongly advise to use the value computed and assigned internally. This value is computed in the computeGenomeWideScores function.

Value

Returns the value assigned to the minPWMScore slot in a genomicProfiles object.

Author(s)

Patrick C. N. Martin <p.martin@essex.ac.uk>

References

Examples

```r
# Data extraction
data(ChIPanalyserData)
# path to Position Frequency Matrix
PFM <- file.path(system.file("extdata", package="ChIPanalyser"), "BEAF-32.pfm")
# As an example of genome, this example will run on the Drosophila genome

if(!require("BSgenome.Dmelanogaster.UCSC.dm6", character.only = TRUE)){
  if (!requireNamespace("BiocManager", quietly=TRUE))
    install.packages("BiocManager")
  BiocManager::install("BSgenome.Dmelanogaster.UCSC.dm6")
}
library(BSgenome.Dmelanogaster.UCSC.dm6)
DNASequenceSet <- getSeq(BSgenome.Dmelanogaster.UCSC.dm6)
# Building data objects
GPP <- genomicProfiles(PFM=PFM, PFMFormat="JASPAR")

# Computing Genome Wide
GenomceWide <- computeGenomeWideScores(DNASequenceSet = DNASequenceSet,
                                           genomicProfiles = GPP)
minPWMScore(GenomceWide)

## If used before computeGenomeWidePWMScore, will return NULL
```

minPWMScore-methods  ~~ Methods for Function minPWMScore ~~

**Description**

Accessor for `minPWMScore`

**Methods:**

- `minPWMScore(object)` Minimum PWM score computed during the `computeGenomeWideScores` step.

naturalLog  ~~ Accessor method the naturalLog slot in a parameterOptions object.~~

**Description**

Accessor method the `naturalLog` slot in a `parameterOptions` object.

**Usage**

- `naturalLog(object)`
Arguments

object object is parameterOptions object.

Details

During the computation of a Postion Weight Matrix, the Position Probability Matrix (derived from a Position Frequency Matrix) is log transformed. This parameter provides which "log transform" will be used. If TRUE, the Natural Log will be used (ln). If FALSE, log2 will be used. We strongly encourage to set values when building a parameterOptions object.

Value

Returns the value assigned to the naturalLog slot in a parameterOptions object.

Author(s)

Patrick C.N. Martin <p.martin@essex.ac.uk>

References


Examples

# Loading data
data(ChIPanalyserData)

# Building data objects
GPP <- parameterOptions(naturalLog=TRUE)
# Setting new Value for naturalLog
naturalLog(GPP)
Setter method for the naturalLog slot in a parameterOptions object.

Description

Setter method for the naturalLog slot in a parameterOptions object.

Usage

naturalLog(object) <- value

Arguments

object object is a parameterOptions object.
value value is a logical value that will determine if the natural log or log2 should be used for the computation of the Position Weight Matrix.

Details

During the computation of a Position Weight Matrix, the Position Probability Matrix (derived from a Position Frequency Matrix) is log transformed. This parameter provides which "log transform" will be used. If TRUE, the Natural Log will be used (ln). If FALSE, log2 will be used. We strongly encourage to set values when building a parameterOptions object.

Value

Returns parameterOptions object with an updated value for the naturalLog slot.

Author(s)

Patrick C.N. Martin <p.martin@essex.ac.uk>

References


Examples

# Loading data
data(ChIPanalyserData)

#Building data objects
OPP <- parameterOptions(naturalLog=TRUE)
#Setting new Value for naturalLog
naturalLog(OPP) <- FALSE
naturalLog<-methods  ~~ Methods for Function naturalLog<-  ~~

Description

Setter method for the naturalLog slot in a parameterOptions object.

Methods:

naturalLog(object)<-value

noiseFilter  Accessor Method for the noiseFilter slot in a parameterOptions object

Description

Accessor Method for the noiseFilter slot in a parameterOptions object

Usage

noiseFilter(object)

Arguments

object  object is parameterOptions object

Details

Noise filtering method that should be used on ChIP-seq data. Four methods are available: Zero, Mean, Median and Sigmoid. Zero removes all ChIP-seq scores below zero, mean under the mean score, median under median score and sigmoid assigns a weight to each score based on a logistic regression curve. Midpoint is set at 95 95 quantile of ChIP-seq scores. Below midpoint will receive a score between 0 and 1, everything above will receive a score between 1 and 2

Value

Returns the value assigned to the noiseFilter slot in a parameterOptions object.

Author(s)

Patrick C. N. Martin <p.martin@essex.ac.uk>

References

Examples

```r
# Loading data
data(ChIPanalyserData)

# Building data objects
GPP <- parameterOptions(noiseFilter="sigmoid")
# Setting new Value for noiseFilter
noiseFilter(GPP)
```

Description

Accessor method for `noiseFilter`

Methods:

- `noiseFilter(object)` Noise Filter that will be applied to ChIP scores

Description

Setter Method for the `noiseFilter` slot in a `parameterOptions` object

Usage

```r
noiseFilter(object) <- value
```

Arguments

- `object` object is `parameterOptions` object
- `value` value is the value to be assigned to the `noiseFilter` slot (zero - mean - median - sigmoid)

Details

Noise filtering method that should be used on ChIP-seq data. Four methods are available: Zero, Mean, Median and Sigmoid. Zero removes all ChIP-seq scores below zero, mean under the mean score, median under median score and sigmoid assigns a weight to each score based on a logistic regression curve. Mid point is set at 95 95 quantile of ChIP-seq scores. Below midpoint will receive a score between 0 and 1, everything above will receive a score between 1 and 2
Value

Returns the value assigned to the noiseFilter slot in a `parameterOptions` object.

Author(s)

Patrick C. N. Martin <p.martin@essex.ac.uk>

References


Examples

```r
# Loading data
data(ChIPanalyserData)

# Building data objects
GPP <- parameterOptions(noiseFilter="sigmoid")
# Setting new Value for noiseFilter
noiseFilter(GPP) <-"zero"
```

Description

Setter method for noiseFilter

Methods:

- `noiseFilter(object)<-value` Noise Filter that will be applied to ChIP scores

noOfSites

**Accessor Method for the noOfSites slot in a parameterOptions object**

Description

Accessor Method for the noOfSites slot in a `parameterOptions` object

Usage

```r
noOfSites(object)
```
Arguments

object object is parameterOptions object

Details

While computing Position Weight Matricies (PWM) from Position Frequency Matricies (PFM), it is possible to restrict the number of sites that will be used to compute the PWM. The default is set at "all". In this case, all sites will be used to compute the PWM.

Value

Returns the value assigned to the noOfSites slot in a parameterOptions object.

Author(s)

Patrick C. N. Martin <p.martin@essex.ac.uk>

References


Examples

# Loading data
data(ChIPanalyserData)

#Building data objects
GPP <- parameterOptions(noOfSites="all")
#Setting new Value for naturalLog
noOfSites(GPP)

---

noOfSites-methods ~ Methods for Function noOfSites ~~

Description

~~ Methods for function noOfSites ~~

Methods:

signature(object = "parameterOptions")
noOfSites<-  

Setter Method for the noOfSites slot in a parameterOptions object.

Description

Setter Method for the noOfSites slot in a parameterOptions object.

Usage

noOfSites(object) <- value

Arguments

object object is a parameterOptions object.
value value is a positive integer that will be assigned to the noOfSites slot.

Details

While computing Position Weight Matrices (PWM) from Position Frequency Matrices (PFM), it is possible to restrict the number of sites that will be used to compute the PWM. The default is set at "all". In this case, all sites will be used to compute the PWM.

Value

Returns a parameterOptions object with an updated value for the noOfSites slot.

Author(s)

Patrick C.N. Martin <p.martin@essex.ac.uk>

References


Examples

# Loading data
data(ChIPanalyserData)

#Building data objects
GPP <- parameterOptions(noOfSites=0)
#Setting new Value for naturalLog
noOfSites(GPP) <- 8
Description

Setter method for `noOfSites`

Methods:

```r
noOfSites(object) <- "all"
noOfSites(object) <- value
```

nos-class

Class "nos"

Description

Virtual class to handle Number of Sites

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

No methods defined with class "nos" in the signature.

Author(s)

Patrick C. N. Martin <pm16057@essex.ac.uk>

References


Examples

```r
showClass("nos")
```
**parameterOptions**  

**parameter Options object**

### Description

`parameterOptions` is an object used to store and parse the various parameters needed throughout this analysis pipeline.

### Usage

```r
parameterOptions(ploidy = 2, boundMolecules = 1000, backgroundSignal = 0, maxSignal = 1, lociWidth = 20000, chipMean = ..., stepSize = 10, removeBackground = 0, noiseFilter = "Zero", naturalLog = TRUE, noOfSites = "all", PWMThreshold = 0.7, strandRule = "max", whichstrand = "+-", PWMpseudocount = 1, lambdaPWM = 1)
```

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ploidy</code></td>
<td>A numeric Value describing the ploidy of the organism. Default: 2</td>
</tr>
<tr>
<td><code>boundMolecules</code></td>
<td>A vector (or single value) containing the number of bound Molecules (bound Transcription Factors): Default: 1000</td>
</tr>
<tr>
<td><code>backgroundSignal</code></td>
<td>A numeric value describing the ChIP-seq background Signal (average signal from real ChIP seq data). Default: 0</td>
</tr>
<tr>
<td><code>maxSignal</code></td>
<td>A numeric value describing the highest ChIP-seq signal (from real ChIP-seq data). Default: 1</td>
</tr>
<tr>
<td><code>lociWidth</code></td>
<td>A numeric value describing the width of the bins used to split ChIP profiles parsed to <code>processingChIP</code>. Default = 20000</td>
</tr>
<tr>
<td><code>chipMean</code></td>
<td>A numeric value describing the mean width of a ChIP- seq peak: Default:200</td>
</tr>
<tr>
<td><code>chipSd</code></td>
<td>A numeric value describing the standard deviation of ChIP-seq peaks. Default: 200</td>
</tr>
<tr>
<td><code>chipSmooth</code></td>
<td>A numeric value describing the width of the window used to smooth Occupancy profiles into ChIP profiles. Default:250</td>
</tr>
<tr>
<td><code>stepSize</code></td>
<td>A numeric value describing the step Size (in base pairs) between each ChIP-seq score. Default:10 (Scored every 10 base pairs)</td>
</tr>
<tr>
<td><code>removeBackground</code></td>
<td>A numeric value describing the value at which score should be removed. Default:0 (If negative scores then remove)</td>
</tr>
<tr>
<td><code>noiseFilter</code></td>
<td>A character string of one of the following: Zero, Mean, Median, or Sigmoid. Noise filter that will be applied to the ChIP Score during the processingChIP step.</td>
</tr>
<tr>
<td><code>naturalLog</code></td>
<td>A logical value describing if natural Log will be used to compute the PWM (if FALSE then log2 will be used). Default: TRUE</td>
</tr>
<tr>
<td><code>noOfSites</code></td>
<td>A Positive integer describing number of sites (in base pair) should be used from the PFM to compute PWM. Default =0 (Full width of binding site will be used when set to 0)</td>
</tr>
</tbody>
</table>
**Details**

ChipAnalyser requires a lot of parameters. `parameterOptions` was created with the intent of storing and parsing these numerous arguments to the different functions. All parameters in this object are optional although strongly recommend. Some parameters are extracted and updated from function along the pipeline e.g. `maxSignal` and `backgroundSignal` are extracted during the `processingChIP` step. These parameters will be automatically parsed. If you do not which to use them (or any other parameter) simply parse a new `parameterOptions` object with your desired parameters.

**Value**

Returns a `parameterOptions` with updated values.

**Author(s)**

Patrick C. N. Martin <p.martin@essex.ac.uk>

**References**


**See Also**

`genomicProfiles`

**Examples**

```r
# parameterOptions(ploidy = 2, boundMolecules = 1000, backgroundSignal = 0,
maxSignal = 1, lociWidth = 20000, chipMean = 200, chipSd = 200,
chipSmooth = 250, stepSize = 10, removeBackground = 0, noiseFilter = "zero",
naturalLog = TRUE, noOfSites = "all", PWMThreshold = 0.7,
strandRule = "max", whichstrand = "+-", PWMpseudocount = 1,
lambdaPWM = 1)
```
parameterOptions-class

Class "parameterOptions"

Description

parameterOptions is an object used to store and parse the various parameters needed throughout this analysis pipeline.

Objects from the Class

Objects can be created by calls of the form parameterOptions(ploidy, boundMolecules, backgroundSignal, maxSignal, lociWidth, chipMean, chipSd, chipSmooth, stepSize, noiseFilter, removeBackground, lambdaPWM, PWMpseudocount, naturalLog, noOfSites, PWMThreshold, strandRule, whichstrand).

Slots

ploidy: Object of class "numeric": A numeric Value describing the ploidy of the organism. Default: 2

boundMolecules: Object of class "vector": A vector (or single value) containing the number of bound Molecules (bound Transcription Factors): Default: 1000

backgroundSignal: Object of class "numeric": A numeric value describing the ChIP-seq background Signal (average signal from real ChIP seq data). Default: 0

maxSignal: Object of class "numeric": A numeric value describing the highest ChIP-seq signal (from real ChIP-seq data). Default: 1

lociWidth: Object of class "numeric": A numeric value describing bin size when splitting ChIP seq scores). Default: 20 000

chipMean: Object of class "numeric": A numeric value describing the mean width of a ChIP-seq peak. Default: 150

chipSd: Object of class "numeric": A numeric value describing the standard deviation of ChIP-seq peaks. Default: 150

chipSmooth: Object of class "vector": A numeric value describing the width of the window used to smooth Occupancy profiles into ChIP profiles. Default:250

stepSize: Object of class "numeric": A numeric value describing the step Size (in base pairs) between each ChIP-seq score. Default:10 (Scored every 10 base pairs)

removeBackground: Object of class "numeric": A numeric value describing the value at which score should be removed. Default:0 (If negative scores then remove)

noiseFilter: Object of class "character" Describes noiseFilter method applied to ChIP scores

PWMThreshold: Object of class "numeric": Threshold at which PWM Score should be selected (only sites above threshold will be selected - between 0 and 1)

strandRule: Object of class "character": "mean", "max" or "sum" will determine how strand should be handle for computing PWM Scores. Default: "max"
whichStrand: Object of class "character": "+","-" or "+-" on which strand should PWM Score be computed. Default: "+-

lambdaPWM: Object of class "vector" A vector (or single value) containing values for lambdaPWM
Default: 1

naturalLog: Object of class "logical": A logical value describing if natural Log will be used to compute the PWM (if FALSE then log2 will be used). Default: TRUE

noOfSites: Object of class "nos" A Positive integer describing number of sites (in base pair) should be used from the PFM to compute PWM. Default = 0 (Full width of binding site will be used when set to 0)

PWMpseudocount: Object of class "numeric": A numeric value describing a PWMpseudocount for PWM computation. Default: 1

paramTag: Object of class "character"~Internal~

Methods

.paramTag signature(object = "parameterOptions"): ...
.paramTag<- signature(object = "parameterOptions", value = "character"): ...
.ZeroBackground signature(object = "parameterOptions"): ...
.ZeroBackground<- signature(object = "parameterOptions", value = "vector"): ...
.backgroundSignal signature(object = "parameterOptions"): ...
.backgroundSignal<- signature(object = "parameterOptions", value = "numeric"): ...
.boundMolecules signature(object = "parameterOptions"): ...
.boundMolecules<- signature(object = "parameterOptions", value = "vector"): ...
.chipMean signature(object = "parameterOptions"): ...
.chipMean<- signature(object = "parameterOptions", value = "numeric"): ...
.chipSd signature(object = "parameterOptions"): ...
.chipSd<- signature(object = "parameterOptions", value = "numeric"): ...
.chipSmooth signature(object = "parameterOptions"): ...
.chipSmooth<- signature(object = "parameterOptions", value = "vector"): ...
.initialize signature(.Object = "parameterOptions"): ...
.lambdaPWM signature(object = "parameterOptions"): ...
.lambdaPWM<- signature(object = "parameterOptions", value = "vector"): ...
.lociWidth signature(object = "parameterOptions"): ...
.lociWidth<- signature(object = "parameterOptions", value = "numeric"): ...
.maxSignal signature(object = "parameterOptions"): ...
.maxSignal<- signature(object = "parameterOptions", value = "numeric"): ...
.naturalLog signature(object = "parameterOptions"): ...
.naturalLog<- signature(object = "parameterOptions", value = "logical"): ...
.noiseFilter signature(object = "parameterOptions"): ...
.noiseFilter<- signature(object = "parameterOptions", value = "character"): ...
Author(s)

Partick C. N. Martin <p.martin@essex.ac.uk>

References


See Also

genomicProfiles

Examples

showClass("parameterOptions")
PFMFormat

Accesor method for the PFMFormat slot in a genomicProfiles object

Description

Accesor method for the PFMFormat slot in a genomicProfiles object

Usage

PFMFormat(object)

Arguments

object object is a genomicProfiles object

Details

If loading a PositionFrequencyMatrix from a file, the format of the file should be specified. Default is raw. Please keep in mind that this argument is used when parsing the PositionFrequencyMatrix file. IF this argument is changed after building the genomicProfiles with a PositionFrequencyMatrix file, this will not influence the parsing of the file. PFMFormat can be one of the following: "raw", "transfac", "JASPAR" or "sequences"

Value

Returns the value assigned to the PFMFormat slot a genomicProfiles

Author(s)

Patrick C. N. Martin <pm16057@essex.ac.uk>

References


Examples

# Loading data
data(ChIPanalyserData)
#Loading PFM files
PFM <- file.path(system.file("extdata",package="ChIPanalyser"),"BEAF-32.pfm")
#Building data objects
##### THIS IS THE PREFERRED METHOD FOR SETTING PFMFormat
GPP <- genomicProfiles(PFM=PFM,PFMFormat="JASPAR")
# Setting New value for PFMFormat

PFMFormat(GPP)

---

**Methods**

**PFMFormat(object)**

---

## Description

Accesor method for the PFMFormat slot in a genomicProfiles object

## Methods:

PFMFormat(object)

---

## Description

Setter method for the PFMFormat slot in a genomicProfiles object

## Usage

PFMFormat(object) <- value

## Arguments

- **object**
  - object is a genomicProfiles object
- **value**
  - value is character string of one of the following: "raw","transfac","JASPAR" or "sequences". If loading a PositionFrequencyMatrix from a file, the format of the file should specified. Default is JASPAR.

## Details

If loading a PositionFrequencyMatrix from a file, the format of the file should be specified. Default is JASPAR. Please keep in mind that this argument is used when parsing the PositionFrequencyMatrix file. IF this argument is changed after building the genomicProfiles with a PositionFrequencyMatrix file, this will not influence the parsing of the file.

## Value

Returns a genomicProfiles object with an updated value for the PFMFormat slot.
Author(s)

Patrick C. N. Martin <pm16057@essex.ac.uk>

References


Examples

# Loading data
data(ChIPanalyserData)
#Loading PFM files
PFM <- file.path(system.file("extdata",package="ChIPanalyser"),"BEAF-32.pfm")
#Building data objects
### THIS IS THE PREFERRED METHOD FOR SETTING PFMFormat
GPP <- genomicProfiles(PFM=PFM,PFMFormat="JASPAR")
#Setting New value for PFMFormat
PFMFormat(GPP) <- "JASPAR"

PMFFormat<-methods  ~~ Methods for Function PMFFormat<- ~~

Description

Setter method for the PFMFormat slot in a genomicProfiles object

Methods:

PMFFormat(object)<-value

ploidy  Accessor method for the ploidy slot in a parameterOptions object

Description

Accessor method for the ploidy slot in a parameterOptions object

Usage

ploidy(object)

Arguments

object object is a parameterOptions object
Details

Default value for ploidy is set a 2. It should be mentioned that ChIPanalyser is based on a model that also considers the ploidy of the organism of interest however this only considers simple polyploidy (or haploidy). The model does not consider hybrids such as wheat.

Value

Returns the value assigned to the ploidy slot in a parameterOptions object

Author(s)

Patrick C. N. Martin <pm16057@essex.ac.uk>

References


Examples

# Building parameterOptions object
OPP <- parameterOptions()
# Setting new Value for maxSignal
ploidy(OPP)

Description

Accessor method for the ploidy slot in a parameterOptions object

Methods:

ploidy(object)
ploidy<-  

Setter Method for the ploidy slot in an **parameterOptions** object

**Description**

Setter Method for the ploidy slot in a **parameterOptions** object

**Usage**

ploidy(object)<- value

**Arguments**

<table>
<thead>
<tr>
<th>object</th>
<th>object is a <strong>parameterOptions</strong> object</th>
</tr>
</thead>
</table>
| value        | value is a positive integer that describes the ploidy of the organism of interest.

**Details**

Default value for ploidy is set to 2. It should be mentioned that ChiPanalyser is based on a model that also considers the ploidy of the organism however this only considers simple polyploidy (or haploidy). The model does not consider hybrids such as wheat.

**Value**

Returns a **parameterOptions** object with an updated value for the ploidy slot.

**Author(s)**

Patrick C. N. Martin <pm16057@essex.ac.uk>

**References**


**Examples**

```r
# Building parameterOptions object
OPP <- parameterOptions()
# Setting new Value for maxSignal
ploidy(OPP) <- 2
```
plotOccupancyProfile

Description

plotOccupancyProfile plots the predicted profiles. If provided, this function will also plot ChIP-seq profiles, PWM Scores (or Occupancy), chromatin States, Goodness of Fit estimates and gene information.

Usage

plotOccupancyProfile(predictedProfile, ChIPScore = NULL, chromatinState = NULL, occupancy = NULL, goodnessOfFit = NULL, PWM = FALSE, geneRef = NULL, addLegend = TRUE, ...)

Arguments

predictedProfile

predictedProfile is a either GRanges containing the predicted profiles for one loci, all loci selected for one parameter, or all loci selected for all parameter combinations selected. (see searchSites)

ChIPScore

ChIPScore is a ChIPscore object containing ChIPscore (or a list of numeric values representing ChIP scores (Experimental ChIP))

chromatinState

chromatinState is a GRanges containing accessible DNA sites or chromatin States.

occupancy

occupancy is a GRanges or a genomicProfiles object containing PWM scores and Occupancy (see computeOccupancy)

goodnessOfFit

goodnessOfFit results of the profileAccuracyEstimate function.

PWM

PWM is a logical value that in the case occupancy is provided which of occupancy scores of PWM scores should be plotted. Default set at FALSE

geneRef

geneRef is a GRanges containing gene information on exons, introns, UTR’s, enhancers or any other genetic element to be plotted.
addLegend

`addLegend` is a logical value defining if the legend should be added. The legend will add all elements provided. See details.

... Any other graphical Parameter of the following: `cex`, `cex.lab`, `cex.main`, `densityCS`, `densityGR`, `ylab`, `xlab`, `main`, `colPred`, `colChIP`, `colOccup`, `colCS`, `colGR`, `n_axis_ticks`. See details.

Details

Once the predicted ChIP-seq like profiles have been computed, it is possible to plot these profiles. This function allows to control graphical parameters. In short:

* `col` = color values - exact number of colors or colors that will be used in a colorRampPalette.
* `cex` = font sizes - for text, axis labels and main
* `Density` = fill density for chromatin state and/or geneRef blocks
* `Pred` = predictedProfile
  - `ChIP` = ChIP score (Experimental ChIP data)
  - `CS` = Chromatin States
  - `GR` = Gene reference
  - `Occup` = Occupancy locations

Value

Returns a profile plot with "Occupancy" on the y axis and DNA position on the X-axis.

Author(s)

Patrick C.N. Martin <pcnmartin@gmail.com>

References


Examples

```r
# Data extraction
data(ChIPanalyserData)
# path to Position Frequency Matrix
PFM <- file.path(system.file("extdata", package="ChIPanalyser"),"BEAF-32.pfm")
# As an example of genome, this example will run on the Drosophila genome
if(!require("BSgenome.Dmelanogaster.UCSC.dm6", character.only = TRUE)){
  if (!requireNamespace("BiocManager", quietly=TRUE))
    install.packages("BiocManager")
  BiocManager::install("BSgenome.Dmelanogaster.UCSC.dm6")
}
library(BSgenome.Dmelanogaster.UCSC.dm6)
DNASequenceSet <- getSeq(BSgenome.Dmelanogaster.UCSC.dm6)
# Building data objects
GPP <- genomicProfiles(PFM=PFM, PFMFormat="JASPAR", BPFrequency=DNASequenceSet)
```
# Computing Genome Wide
GenomeWide <- computeGenomeWideScores(DNASequenceSet = DNASequenceSet,
               genomicProfiles = GPP)

#Compute PWM Scores
PWMScores <- computePWMscore(DNASequenceSet = DNASequenceSet,
               genomicProfiles = GenomeWide,
               loci = top, chromatinState = Access)

#Compute Occupancy
Occupancy <- computeOccupancy(genomicProfiles = PWMScores)

#Compute ChIP profiles
chipProfile <- computeChipProfile(loci = top,
               genomicProfiles = Occupancy)

#Plotting Profile
plotOccupancyProfile(predictedProfile=chipProfile,
               ChIPScore = chip,
               chromatinState = Access,
               occupancy = Occupancy,
               geneRef = geneRef)

plotOccupancyProfile(predictedProfile=chipProfile,
               ChIPScore = chip,
               chromatinState = Access,
               occupancy = Occupancy,
               geneRef = geneRef,
               colCS = c("red","blue"),
               densityGR = 60)

---

**plotOptimalHeatMaps**  
*Heat Map of optimal Parameters*

**Description**

plotOptimalHeatMaps will plot heat maps of optimal Parameters and highlight the optimal combination of *lambdaPWM* and *boundMolecules*

**Usage**

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

Arguments

optimalParam optimalParam is a list containing containing optimal matrices (or only one if only one parameter was selected). These matrices are the result of the computeOptimal function.

contour parameter is logical. Should contour lines be plotted?

col col vector of colours to be used for each heat map. If none are specified, rainbow colours will be used. NOTE: colour vector will be recycled if not enough colours are provided.

main main title.

layout layout is either TRUE or FALSE specifying if standard layout should be used or not. If TRUE, each heat map will be plotted on an individual page with a heat map scale of the right side.

overlay overlay is either TRUE or FALSE specifying if an overlay plot should be produced. The overlay plot takes the top 10 percent of best performing parameters per scoring metric and overlays them in a single plot. The resulting plots shows the optimal set of parameters for all metrics combined.

Details

Once the optimal set of Parameters (lambdaPWM and boundMolecules), it is possible to plot the results in the form of a heat map. Each heat map will be plotted on a separate page if layout = TRUE. If layout = FALSE, it is up to the user to define how they wish to layout their heat maps.

Value

Returns a heat map of optimal combinations of lambdaPWM and boundMolecules. The x axis represents the different value assigned to lambda (lambdaPWM) and the y axis represents the different values to boundMolecules (boundMolecules).

Author(s)

Patrick C. N. Martin <pm16057@essex.ac.uk>

References


Examples

# Data extraction
data(ChIPanalyserData)
# path to Position Frequency Matrix
PFM <- file.path(system.file("extdata",package="ChIPanalyser"),"BEAF-32.pfm")
# As an example of genome, this example will run on the Drosophila genome

if(!require("BSgenome.Dmelanogaster.UCSC.dm6", character.only = TRUE)){
  if (!requireNamespace("BiocManager", quietly=TRUE))
    BiocManager::install()

  require(BSgenome.Dmelanogaster.UCSC.dm6, character.only = TRUE)

  require("ChIPanalyser")

  # Extract lambdaPWM and boundMolecules for each scoring metric
  lambdaPWM <- extractLambdaPWM(ChIPanalyserData)
  boundMolecules <- extractBoundMolecules(ChIPanalyserData)

  # Plot heat maps
  plotOptimalHeatMaps(lambdaPWM, boundMolecules, layout=TRUE)
}
install.packages("BiocManager")
BiocManager::install("BSgenome.Dmelanogaster.UCSC.dm6")
}
library(BSgenome.Dmelanogaster.UCSC.dm6)
DNASequenceSet <- getSeq(BSgenome.Dmelanogaster.UCSC.dm6)

# Building data objects
GPP <- genomicProfiles(PFM=PFM,PFMFormat="JASPAR",BPFrequency=DNASequenceSet)

# Computing Optimal set of Parameters
optimalParam <- computeOptimal(genomicProfiles = GPP, DNASequenceSet = DNASequenceSet, ChIPScore = chip, chromatinState = Access, parameterOptions = OPP, parameter = "all", peakMethod="moving_kernel")
plotOptimalHeatMaps(optimalParam)

---

PositionFrequencyMatrix

*Accessor method for the PFM slot in a genomicProfiles object*

**Description**

Accessor method for the PFM slot in a genomicProfiles object

**Usage**

PositionFrequencyMatrix(object)

**Arguments**

- object object is a genomicProfiles object

**Details**

After creating a genomicProfiles object, it is possible to access the Position Frequency Matrix slot. However this slot will be empty if the genomicProfiles object was built using directly a Position Weight Matrix. See genomicProfiles

**Value**

Returns the Position Frequency Matrix (PFM slot) used to compute the PositionWeightMatrix in a genomicProfiles object

**Author(s)**

Patrick C. N. Martin <pm16057@essex.ac.uk>
References


Examples

```r
#Loading data
data(ChIPanalyserData)

#Loading PFM files
PFM <- file.path(system.file("extdata",package="ChIPanalyser"),"BEAF-32.pfm")

#Building genomicProfiles object
GPP<-genomicProfiles(PFM=PFM,PFMFormat="JASPAR")
# Accessing Slot
PositionFrequencyMatrix(GPP)
```

Description

Accessor method for the PFM slot in a genomicProfiles object

Methods:

```r
PositionFrequencyMatrix(object)
```

Setter method for the PFM slot in a genomicProfiles object

```r
PositionFrequencyMatrix<- (object,value)
```

Arguments

- `object`: object is a genomicProfiles object
- `value`: value can be of two forms. Either a matrix in the form of a Position Frequency Matrix or a path/to/file character string.
Details

The Position Frequency Matrix is one of the fundamental objects that needs to be supplied to a

**genomicProfiles**. If after building a **genomicProfiles**, only the Position Frequency Matrix needs
to be modified then it is possible to manually update the value of this matrix using the function
above. There are two options for the type of data that may be supplied to the PFM slot: a matrix in
the form of a Position Frequency Matrix (matrix with four rows - one for each base pair (ACTG)
and a number of columns equal to the number of sites in the binding site), or it is possible (also
recommended) to provide a path to the file containing the Position Frequency Matrix. This Position
Frequency Matrix file may come in multiple form such as RAW, Transfac or JASPAR. **WARNING:**
if a genomicProfiles object has already been created and only the PFM is supplied/updated , then
the PFM slot will automatically be updated as well.

Value

Returns a **genomicProfiles** with an updated PFM slot (as described above this will lead to an
updated PositionWeightMatrix).

Author(s)

Patrick C. N. Martin <pm16057@essex.ac.uk>

References


Examples

```r
#Loading data
data(ChIPanalyserData)
#Loading PFM files
PFM <- file.path(system.file("extdata",package="ChIPanalyser"),"BEAF-32.pfm")
#Building genomicProfiles object
# NOT ADVISED!!!! PLEASE PARSE PFM AND PFMFormat together
GPP<-genomicProfiles(PFMFormat = "JASPAR")
#Setting PFM
PositionFrequencyMatrix(GPP) <- PFM
```

```
PositionFrequencyMatrix<--methods

`~ Methods for Function PositionFrequencyMatrix<-- ~~`

Description

Setter method for the PFM slot in a **genomicProfiles** object

Methods:

- PositionFrequencyMatrix(object)<-"path/to/file/
- PositionFrequencyMatrix(object)<-value
**PositionWeightMatrix**    
*Accessor Method for the PWM slot in a genomicProfiles object*

**Description**

Accessor Method for the PWM slot in a genomicProfiles object

**Usage**

PositionWeightMatrix(object)

**Arguments**

object object is a genomicProfiles

**Details**

After creating a genomicProfiles object, it is possible to access the Position Weight Matrix stored in this slot. This slot should always contain something. This slot is either supplied by user or directly computed from a Position Frequency Matrix when supplied.

**Value**

Returns a matrix in the form of a Position Weight Matrix

**Author(s)**

Patrick C. N. Martin <pm16057@essex.ac.uk>

**References**


**Examples**

```r
#Loading data
data(ChIPanalyserData)
#Loading PFM files
PFM <- file.path(system.file("extdata",package="ChIPanalyser"),"BEAF-32.pfm")
#Building genomicProfiles object
GPP<-genomicProfiles(PFM=PFM,PFMFormat="JASPAR")
# Accessing Slot
PositionWeightMatrix(GPP)
```
Description

Accessor Method for the PWM slot in a genomicProfiles object

Methods:

PositionWeightMatrix(object)

Setter Method for the PositionWeightMatrix slot in a genomicProfiles object

Description

Setter Method for the PositionWeightMatrix slot in a genomicProfiles object

Usage

PositionWeightMatrix(object) <- value

Arguments

object object is a genomicProfiles object
value value is a matrix in the form of a Position Weight Matrix.

Details

If a Position Weight Matrix is readily available, it is possible to directly assign this matrix to the PWM slot. However, this is only possible if a genomicProfiles object has already been created. In that case, we advise to first create a genomicProfiles object. It should be noted that this Position Weight Matrix will be automatically computed from a Position Frequency Matrix. If no Position Frequency Matrix are available, then a Position Weight Matrix can be directly assigned to this slot.

Value

Returns a genomicProfiles object with an updated value for the PWM slot

Author(s)

Patrick C. N. Martin <pm16057@essex.ac.uk>
References


Examples

# Building genomicProfiles object
GPP <- genomicProfiles()
# Setting PWM to PositionWeightMatrix slot
PWM <- matrix(runif(32,-10,20), ncol=8)
rownames(PWM) <- c("A","C","T","G")
PositionWeightMatrix(GPP) <- PWM

Description

Setter Method for the PositionWeightMatrix slot in a genomicProfiles object

Methods:

PositionWeightMatrix(object)<-value

preprocessingChIP  Pre-processing ChIP-seq data

Description

preprocessingChIP will process and extract ChIP scores at a set of loci of interest.

Usage

preprocessingChIP(profile, loci=NULL, reduce=NULL,
                  peaks=NULL, chromatinState=NULL, parameterOptions=NULL,
                  cores=1)
Arguments

profile: profile is a path to a UCSC format file, a GRanges or data frame. The input data should contain 4 columns: chromosome, start, end and score.

loci: loci is GRanges describing the loci at which ChIP scores should be extracted. If NULL, a set of Loci will be extracted from profile. The data provided will then be split into bins of width equal to lociWidth (Default 20kbp). Default=NULL

reduce: reduce is the top regions to select based on the mean ChIP score. If peaks are provided, regions overlapping with known peaks will be selected based on highest ChIP score. If NULL, all regions will be considered. Default=NULL

parameterOptions: parameterOptions is a parameterOptions object containing ChIP parameters to be parsed for ChIP score extraction. If NULL, parameterOptions will be built internally with default ChIP extraction parameters (see chipSmooth, chipSd, and chipMean). Default=NULL

peaks: peaks is a path to UCSC format file or a GRanges object containing location of ChIP peaks. Default=NULL

chromatinState: chromatinState is a GRanges containing Accessible DNA or chromatin States. If provided, regions will be selected only if they contain accessible DNA. Default=NULL

cores: cores is the number of cores used to extract ChIP scores. Default = 1

Details

When using computeOptimal, it is required to supply real ChIP data in order to have a point of comparison. The correlation and MSE Scores are computed based on how well the model fits biological data. processingChIP will extract this data from ChIP data at loci of interest. When using the reduce option, this function will only select the top regions based on peak height or mean ChIP score. processingChIP will also extract maxSignal and backgroundSignal from ChIP data and parse it to a parameterOptions object.

Value

Returns a ChIPScore object containing extracted (and normalised) ChIP scores, the loci of interest and newly extracted Parameters (e.g. maxSignal)

Author(s)

Patrick C.N. Martin <pcnmartin@gmail.com>

References

profileAccuracyEstimate

Examples

# Data extraction
data(ChIPanalyserData)

## Extracting ChIP scores at loci of interest
ChIP<-processingChIP(profile=chip, loci=top)

profileAccuracyEstimate

Estimating Accuracy of predicted Profiles

Description

profileAccuracyEstimate will compare the predicted ChIP-seq-like profile to real ChIP-seq data and return a set of metrics describing how accurate the predicted model is compared to real data.

Usage

profileAccuracyEstimate(genomicProfiles,ChIPScore, parameterOptions=NULL,method="all",cores=1)

Arguments

genomicProfiles
genomicProfiles is the result of computeChIPProfile

ChIPScore
ChIPScore is the result of processingChIP. Extracted/Normalised experimental ChIP scores.

parameterOptions
parameterOptions is a parameterOptions object for parameter specification.

method
method is the method that will be used to assess model quality against ChIP-seq data. Method can be one of the following: pearson, spearman, kendall, ks, geometric,fscore, MSE,or all. Fscore contains f-score, precision,recall, MCC, Accuracy and AUC ROC.

cores
cores is the number of cores used to extract ChIP scores. Default = 1

Details

In order to assess the quality of the model against experimental ChIP-seq data, ChIPanalyser offers a wide range of method to choose from. These methods are also used when computing optimal parameters.

Value

Returns list of goodness of fit metrics for each loci and each parameter selected.
Author(s)

Patrick C. N. Martin <pm16057@essex.ac.uk>

References


Examples

# Data extraction
data(ChIPanalyserData)
# path to Position Frequency Matrix
PFM <- file.path(system.file("extdata", package="ChIPanalyser"), "BEAF-32.pfm")
# As an example of genome, this example will run on the Drosophila genome
if(!require("BSgenome.Dmelanogaster.UCSC.dm6", character.only = TRUE)){
  if (!requireNamespace("BiocManager", quietly=TRUE))
    install.packages("BiocManager")
  BiocManager::install("BSgenome.Dmelanogaster.UCSC.dm6")
}
library(BSgenome.Dmelanogaster.UCSC.dm6)
DNASequenceSet <- getSeq(BSgenome.Dmelanogaster.UCSC.dm6)
# Building genomicProfiles object
GPP <- genomicProfiles(PFM=PFM, PFMFormat="JASPAR", BPFrequency=DNASequenceSet)

# Computing Genome Wide
GenomeWide <- computeGenomeWideScore(genomicProfiles = GPP,
                                      DNASequenceSet = DNASequenceSet)

# Compute PWM Scores
PWMScores <- computePWMScore(genomicProfiles = GenomeWide,
                               DNASequenceSet = DNASequenceSet, loci = top, chromatinState = Access)

# Compute Occupancy
Occupancy <- computeOccupancy(genomicProfiles = PWMScores)

# Compute ChIP profiles
chipProfile <- computeChIPProfile(genomicProfiles = Occupancy, loci = top)

# Estimating accuracy estimate
AccuracyEstimate <- profileAccuracyEstimate(genomicProfiles = chipProfile,
                                          ChIPScore = chip,
                                          occupancyProfileParameters = OPP)
Description

Accessor method for profiles in a genomicProfiles object

Methods:

profiles(object) Computed PWM scores, Occupancy or ChIP-seq like profiles for loci of interest and parameter combination of interest.

PWMpseudocount

Accesser Method for a PWMpseudocount slot in a parameterOptions

Description

Accessor Method for a PWMpseudocount slot in a parameterOptions

Usage

PWMpseudocount(object)

Arguments

object object is a parameterOptions object.

Details

In the context of Position Weight Matricies, the pseudocount is used to avoid 0 probabilities during the transformation of Position Frequency Matrix to a Position Probability Matrix and finally to a Postion Weight Matrix. It is essentially a sample correction that is added in the case of small sample size. The effect of the base pair to which a pseudocount was assigned will not influence the model nor will create mathematical issues such as infinities or zero division. Default is set at 1.

Value

Returns the value assigned to a PWMpseudocount slot in a parameterOptions object

Author(s)

Patrick C. N. Martin <pm16057@essex.ac.uk>
References


Examples

# Loading data
data(ChIPanalyserData)

#Building data objects
GPP <- parameterOptions(PWMpseudocount=0)
#Accessing slot value
PWMpseudocount(GPP)

PWMpseudocount-methods

~~ Methods for Function PWMpseudocount ~~

Description

Accessor Method for a PWMpseudocount slot in a parameterOptions

Methods:

PWMpseudocount(object)

PWMpseudocount<-  

Setter Method for the pseudocount slot in a parameterOptions object

Description

Setter Method for the pseudocount slot in a parameterOptions object

Usage

PWMpseudocount(object) <- value

Arguments

object object is a parameterOptions object
value value is a numeric value that will be assigned to the pseudocount slot. Default is set at 1
Details

In the context of Position Weight Matricies, the pseudocount is used to avoid 0 probabilities during the transformation of Position Frequency Matrix to a Position Probability Matrix and finally to a Position Weight Matrix. It is essentially a sample correction that is added in the case of small sample size. The effect of the base pair to which a pseudocount was assigned will not influence the model nor will create mathematical issues such as infinities or zero division.

Value

Returns a parameterOptions object with an updated value for the pseudocount slot.

Author(s)

Patrick C. N. Martin <pm16057@essex.ac.uk>

References


Examples

# Loading data
data(ChIPanalyserData)

#Building data objects
GPP <- parameterOptions( PWMpseudocount=0)
#Setting Value for new PWMpseudocount
PWMpseudocount(GPP) <- 1
PWMThreshold

Accessor method for the PWMThreshold slot in a parameterOptions object

Description

Accessor method for the PWMThreshold slot in a parameterOptions object

Usage

PWMThreshold(object)

Arguments

object object is a parameterOptions object

Details

The computePWMScore function requires a so-called PWM Threshold. This threshold represents the Threshold at which PWM Score should be selected. The PWMThreshold is a positive numeric value (between 0 and 1). If set at 0, all sites will be selected. If set at 0.7 (Default value), then 70% of PWM Score (and by extension binding sites) will be IGNORED. The top 30% will be selected.

Value

Returns the value assigned to the PWMThreshold slot in a parameterOptions object

Author(s)

Patrick C. N. Martin <pm16057@essex.ac.uk>

References


Examples

# Loading data
data(ChIPanalyserData)

# Building data objects
GPP <- parameterOptions(PWMThreshold=0.7)

# Accessing Value for PWMThreshold
PWMThreshold(GPP)
**Description**

Accessor method for the PWMThreshold slot in a parameterOptions object

**Methods:**

PWMThreshold(object)

**PWMThreshold<-**

Setter Method for the PWMThreshold slot in a parameterOptions object

**Description**

Setter Method for the PWMThreshold slot in a parameterOptions object

**Usage**

PWMThreshold(object) <- value

**Arguments**

object is a parameterOptions object

value is a numeric value (between 0 and 1) to be assigned to the PWMThreshold slot in parameterOptions object. Default is set at 0.7

**Details**

The computePWMscore function requires a so-called PWM Threshold. This threshold represents the Threshold at which PWM Score should be selected. The PWMThreshold is a positive numeric value (between 0 and 1). If set at 0, all sites will be selected. If set at 0.7 (Default value), then 70% of PWM Score (and by extension binding sites) will be IGNORED. The top 30% will be selected.

**Value**

Returns parameterOptions object with an updated value for the PWMThreshold slot

**Author(s)**

Patrick C. N. Martin <pm16057@essex.ac.uk>
References


Examples

# Loading data
data(ChIPanalyserData)

#Building data objects
GPP <- parameterOptions(PWMThreshold=0.7)
#Setting Value for new PWMThreshold
PWMThreshold(GPP) <- 0.8

removeBackground(object) <- value

removeBackground accessor method for the removeBackground slot in a parameterOptions object

Details

A numeric value describing a threshold at which Occupancy signals must be removed (Default is set at 0). The removal of Occupancy signals will occur when computing computeOccupancy function.
**Value**

Returns the value assigned to the `removeBackground` slot in a `parameterOptions` object.

**Author(s)**

Patrick C. N. Martin <pm16057@essex.ac.uk>

**References**


**Examples**

#Building parameterOptions object
OPP <- parameterOptions()
#Accessing Value for removeBackground
removeBackground(OPP)
Arguments

object object is an parameterOptions object
value value is positive numerical value to be assigned to the removeBackground slot in a parameterOptions object. Default is set at 0.

Details

A numeric value describing a threshold at which Occupancy signals must be removed (Default is set at 0). The removal of Occupancy signals will occur when computing computeOccupancy (see computeOccupancy function)

Value

Returns an parameterOptions object with an updated value for the removeBackground slot

Author(s)

Patrick C. N. Martin <pm16057@essex.ac.uk>

References


Examples

#Building parameterOptions object
OPP <- parameterOptions()
#Setting new Value for removeBackground
removeBackground(OPP) <- 0.1

Description

Setter Method for the removeBackground slot in a parameterOptions object

Methods:

removeBackground(object)<-value
scores

Accessor Method for the scores slot in a ChIPScore object

**Description**

Setter Method for the scores slot in a ChIPScore object

**Usage**

scores(object)

**Arguments**

object object is ChIPScore object

**Details**

When using the processingChIP, this functions will return a name list of normalised ChIP scores at loci of interest. This functions enables you to extract those scores from the ChIPScore object.

**Value**

Returns the value assigned to the scores slot in a ChIPScore object.

**Author(s)**

Patrick C. N. Martin <p.martin@essex.ac.uk>

**References**


**Examples**

```r
# Loading data
data(ChIPanalyserData)

chip<-processingChIP(chip,top)
str(scores(chip))
```
scores-methods  

--- Methods for Function scores ---

Description

Accessor method for scores slot in a ChIPScore object.

Methods:

scores(object) Extracted and normalised ChIP scores at loci of interest.

searchSites  

Searching function for Sites above threshold and predicted ChIP-seq Profiles

Description

searchSites is function enabling quick extraction and search for parameter combinations and/or loci in any genomicProfiles object from computeOccupancy onwards.

Usage

searchSites(Sites, lambdaPWM = "all", BoundMolecules = "all", Locus = "all")

Arguments

<table>
<thead>
<tr>
<th>Sites</th>
<th>Sites is either a genomicProfiles or the result of computeOptimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>lambdaPWM</td>
<td>lambdaPWM is a numeric vector describing the ScalingFactors that should be searched within Sites.</td>
</tr>
<tr>
<td>BoundMolecules</td>
<td>BoundMolecules is a numeric vector describing the BoundMolecules that should be searched within Sites.</td>
</tr>
<tr>
<td>Locus</td>
<td>Locus is a character vector describing the Loci that should be searched within Sites.</td>
</tr>
</tbody>
</table>

Details

When testing numerous combinations of lambdaPWM and boundMolecules on top of many loci, it can become challenging to navigate the large data output searchSites will make searching in this slot a lot easier. If all arguments are left at their default value of "all", then all Parameters will be searched thus returning the full list of Sites above threshold. If a value for lambdaPWM is user provided then only this lambdaPWM will be selected (all boundMolecules and loci will also be selected). searchSites also works on the result of computeOptimal.

Value

Returns object of same time as parsed to this function with only the parameters and/or loci selected.
Author(s)

Patrick C. N. Martin <pm16057@essex.ac.uk>

References


Examples

# Data extraction
data(ChIPanalyserData)
# path to Position Frequency Matrix
PFM <- file.path(system.file("extdata",package="ChIPanalyser"),"BEAF-32.pfm")
# As an example of genome, this example will run on the Drosophila genome
if(!require("BSgenome.Dmelanogaster.UCSC.dm6", character.only = TRUE)){
  if (!requireNamespace("BiocManager", quietly=TRUE))
    install.packages("BiocManager")
  BiocManager::install("BSgenome.Dmelanogaster.UCSC.dm6")
}
l library(BSgenome.Dmelanogaster.UCSC.dm6)
DNASequenceSet <- getSeq(BSgenome.Dmelanogaster.UCSC.dm6)
# Building genomicProfiles object
GPP <- genomicProfiles(PFM=PFM,PFMFormat="JASPAR", BPFrequency=DNASequenceSet)

# Computing Genome Wide
GenomeWide <- computeGenomeWideScore(genomicProfiles = GPP,
DNASequenceSet = DNASequenceSet)

# Compute PWM Scores
PWMScores <- computePWMscore(genomicProfiles = GenomeWide,
DNASequenceSet = DNASequenceSet, loci = top, chromatinState = Access)

# Compute Occupancy
Occupancy <- computeOccupancy(genomicProfiles = PWMScores)
searchSites(Occupancy, ScalingFactor=c(1,4), BoundMolecules = c(1,100),
Locus="eve")

# Compute ChIP profiles
chipProfile <- computeChIPProfile(genomicProfiles=Occupancy,loci=top)
searchSites(chipProfile,ScalingFactor=c(1,4), BoundMolecules = c(1,100),
Locus="eve")

optimalParam <- computeOptimal(genomicProfiles = GPP,
DNASequenceSet = DNASequenceSet,
ChIPScore = chip,
chromatinState = Access,
parameterOptions = OPP,
parameter = "all",

searchSites
Description

setChromatinStates sets chromatin state affinity values to a GRanges object.

Usage

setChromatinStates(population, chromatinStates)

Arguments

- population: Population list containing all individuals and associated parameter. Must contain chromatin state affinity values. See generateStartingPopulation.
- chromatinStates: GRanges object containing chromatin state locations.

Details

Chromatin states can be loaded into R as a GRanges object. Each range represents the extent of a certain chromatin state and the chromatin state type should be assigned to a meta data column called "name". The affinity values names should be set accordingly.

Value

Returns a GRanges object with affinity scores for each chromatin state range. Affinity scores are placed in the DNAAffinity meta data column.

Author(s)

Patrick C.N. Martin

Examples

library(ChIPanalyser)
# Input data
data(ChIPanalyserData)

pop <- 10
params <- c("N", "lambda", "PWMThreshold", paste0("CS", seq(1:11)))
start_pop <- generateStartingPopulation(pop, params)
cs <- setChromatinStates(start_pop,cs)

### show-methods

**show-methods**

~~ Methods for Function show ~~

### Description

Show methods for various objects

### Methods:

- signature(object = "ChIPScore")
- signature(object = "genomicProfiles")
- signature(object = "parameterOptions")

### singleRun

**singleRun**

### Description

singleRun runs ChIPanalyser after optimal parameters have been found by the evolve function.

### Usage

```r
singleRun(indiv, DNAAffinity, genomicProfiles, DNASequenceSet, ChIPScore, fitness="all")
```

### Arguments

- **indiv**
  - Population list containing the top scoring individual. Note that this should be a list of length 1 containing another list with all parameter values.

- **DNAAffinity**
  - GRanges object as outputed by the setchromatinStates.

- **genomicProfiles**
  - genomicProfiles object containing PWM scores and other desired metrics. Note that PWMThreshold, lambda and N will be overwritten using values from indiv.

- **DNASequenceSet**
  - DNA string set object containing DNA sequence of interest.

- **ChIPScore**
  - ChIPScore object as outputed by the processingChIP function.

- **fitness**
  - character string describing which metric should be used to assess fitness and should be one of the following: "geometric", "ks", "MSE", "pearson", "spearman", "kendall", "recall", "precision", "f1score", "MCC", "Accuracy" or "AUC".
Details

Once the genetic algorithm has been optimised, the top individual may be run on its own to get predicted ChIP profiles. The use of this function requires a few extract steps in order to predict ChIP profiles.

First, the index of the top individual should be extracted (see `getHighestFitnessSolutions`). Second, using this index, subset top individual from GA population. Note this should be done using "[]" single bracket notation as, a list of length 1 containing another list with all parameter values is required for the next steps. Yes, this is might seem annoying but the functions were design for list structures... Third, setchromatinStates using the top individual list. This will add chromatin affinity values to your chromatinState GRanges. Use this new chromatinState object as your new chromatinState object. Fourth, parse your indiv list object to `singleRun`.

Value

Return a list with three elements. First element contains a genomicProfiles object with occupancy scores. Second element contains a genomicProfiles object with ChIP profile scores. Third element contains a goodness of fit metrics.

Author(s)

Patrick C.N. Martin <pcnmartin@gmail.com>

Examples

```r
library(ChIPanalyser)
data(ChIPanalyserData)
# See GA vignette for usage
```

---

**splitData**

*Get Training and Testing data from ChIPscore objects*

**Description**

`splitData` splits processed ChIP data into training and testing sets.

**Usage**

```r
splitData(ChIPscore, dist = c(80,20), as.proportion = TRUE)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChIPscore</td>
<td>ChIPscore object as returned by <code>processingChIP</code></td>
</tr>
<tr>
<td>dist</td>
<td>If <code>as.proportion</code> is to <code>TRUE</code>, split the data into desired proportions. Default sets 80% training and 20% testing. If <code>as.proportion</code> is to <code>FALSE</code>, a vector of 4 numeric values describing start and end of training and testing respectively.</td>
</tr>
<tr>
<td>as.proportion</td>
<td>Logical describing if values provided to <code>dist</code> should be treated as % of training and testing or if <code>dist</code> should be considered as start and end of loci selected for training and testing respectively.</td>
</tr>
</tbody>
</table>
**Value**

Returns a named list of ChIPScore objects

* trainingSet = ChIPscore containing training set
* testingSet = ChIPscore containing testing set.

**Author(s)**

Patrick C.N. Martin <pcnmartin@gmail.com

**Examples**

```r
library(ChIPanalyser)
data(ChIPanalyserData)
# See GA vignette for usage
test <- processingChIP(chip,top)
usingDist <- splitData(test, dist = c(50,50),as.proportion = TRUE )
usingIndex <- splitData(test, dist = c(1,2,3,4),as.proportion = FALSE )
```

---

**stepSize**

* **Accessor method of the stepSize slot in parameterOptions object**

**Description**

Accessor method of the stepSize slot in **parameterOptions** object

**Usage**

```r
stepSize(object)
```

**Arguments**

- **object**
  
  object is a **parameterOptions** object.

**Details**

It possible to restrict the size of the ChIP-seq-like profile produced by **computeChIPProfile**. Instead of returning ChIP-seq like score for each base pair, it is possible to skip base pairs and only return the predicted enrichment score for every "n" base pair (n is the value assigned to stepSize). This will reduce the size of the output data (unless step size is very large, this will not affect the accuracy of the model). Default is set at 10 base pairs.

**Value**

Returns the value assigned to the stepSize slot in **parameterOptions**

**Author(s)**

Patrick C. N. Martin <pm16057@essex.ac.uk>
References


Examples

```r
# Building parameterOptions object
OPP <- parameterOptions()
# Setting new Value for maxSignal
stepSize(OPP)
```

---

**Description**

Accessor method of the stepSize slot in `parameterOptions` object

**Methods:**

`stepSize(object)`

---

**stepSize<-**

*Setter Method for the stepSize slot in a parameterOptions*

**Description**

Setter Method for the stepSize slot in a `parameterOptions`

**Usage**

`stepSize(object) <- value`

**Arguments**

| object    | object is a `parameterOptions` object |
| value     | value is a positive numeric value that will be assigned to the stepSize slot in a `parameterOptions` object. Default is set at 10 base pairs. |

**Details**

It possible to restrict the size of the ChIP-seq-like profile produced by `computeChIPProfile`. Instead of returning ChIP-seq like score for each base pair, it is possible to skip base pairs and only return the predicted enrichment score for every "n" base pair (n is the value assigned to stepSize). This will reduce the size of the output data (unless step size is very large, this will not affect the accuracy of the model). Default is set at 10 base pairs.
**Value**

Returns a `parameterOptions` object with an updated value for the `stepSize` slot.

**Author(s)**

Patrick C. N. Martin <pm16057@essex.ac.uk>

**References**


**Examples**

```r
# Building parameterOptions object
OPP <- parameterOptions()
# Setting new Value for maxSignal
stepSize(OPP) <- 20
```

**Description**

Setter Method for the `stepSize` slot in a `parameterOptions` object

**Methods:**

- `stepSize(object) <- value`

**strandRule**

*Accessor Method for the strandRule slot in a parameterOptions object*

**Description**

Accessor Method for the `strandRule` slot in a `parameterOptions` object

**Usage**

`strandRule(object)`
Arguments

object object is a parameterOptions object

Details

When computing the PWM Scores and if whichstrand is set to "+-", strandRule will determine how to handle both strands (one of three options: "mean", "max", "sum"). If set to "mean", the average PWM Score of both strand will be computed. If set to "max", the highest PWM score between each strand will be selected and finally "sum" will sum both score together. Default set at "max"

Value

Returns the value assigned to strandRule slot (one of three options: "mean", "max", "sum") in a parameterOptions object

Author(s)

Patrick C. N. Martin <pm16057@essex.ac.uk>

References


Examples

# Loading data
data(ChIPanalyserData)

#Building data objects
GPP <- parameterOptions(strandRule="max")
#Accesssing Value for strandRule
strandRule(GPP)
**strandRule**<-  

*Setter method for the strandRule slot in a parameterOptions object.*

**Description**

Setter method for the strandRule slot in a parameterOptions object.

**Usage**

```
strandRule(object) <- value
```

**Arguments**

- **object**
  - object is a parameterOptions object
- **value**
  - value is a character string and can be one of the following 'mean', 'max', 'sum'. This will only apply if whichstrand is ‘+-’. Default set at ‘max’

**Details**

When computing the PWM Scores and if whichstrand is set to ‘+-’, strandRule will determine how to handle both strands (one of three options: ‘mean’, ‘max’, ‘sum’). If set to ‘mean’, the average PWM Score of both strand will be computed. If set to ‘max’, the highest PWM score between each strand will be selected and finally ‘sum’ will sum both score together. Default set at ‘max’.

**Value**

Returns a parameterOptions object with an updated value for the strandRule slot.

**Author(s)**

Patrick C. N. Martin <pm16057@essex.ac.uk>

**References**


**Examples**

```
# Loading data
data(ChIPanalyserData)

#Building data objects
GPP <- parameterOptions(strandRule="max")

#Setting New Value for strandRule
strandRule(GPP) <- "mean"
```
whichstrand}

---

**strandRule**<~methods> <~ Methods for Function strandRule<- ~>

**Description**

Setter method for the `strandRule` slot in a `parameterOptions` object.

**Methods:**

`strandRule(object)<-value`

---

**whichstrand**

**Accessor method for the whichstrand slot in a parameterOptions object**

---

**Description**

Accessor method for the `whichstrand` slot in a `parameterOptions` object

**Usage**

`whichstrand(object)`

**Arguments**

- `object` object is a `parameterOptions` object

**Details**

PWM Score may be computed on either the positive strand ("+"), the negative strand ("-"), or on both strands ("+-").

**Value**

Returns on which strand PWM Scores should be computed (whichstrand in a parameterOptions object)

**Author(s)**

Patrick C. N. Martin <pm16057@essex.ac.uk>

**References**

Examples

# Loading data
data(ChIPanalyserData)

# Building data objects
GPP <- parameterOptions( whichstrand="+-")
# Setting New Value for whichstrand
whichstrand(GPP)

whichstrand-methods  ~~ Methods for Function `whichstrand` ~~

Description

Accessor method for the `whichstrand` slot in a parameterOptions object

Methods:

whichstrand(object)

whichstrand<-  ~ ~ Setter method for the whichstrand slot in a parameterOptions object

Description

Setter method for the `whichstrand` slot in a parameterOptions object

Usage

whichstrand(object) <- value

Arguments

object       object is a parameterOptions object
value        value is a character string specifying which strand should be used to compute
PWM Scores. The three available options are the following: "+","-" or "+-". Default is "+-"

Details

PWM Score may be computed on either the positive strand ("+") the negative strand ("-"") or on both strands ("+-").

Value

Returns a parameterOptions object with an updated value for the whichstrand slot
Author(s)

Patrick C. N. Martin <pm16057@essex.ac.uk>

References


Examples

# Loading data
data(ChIPanalyserData)

#Building data objects
GPP <- parameterOptions( whichstrand="+-")
#Setting New Value for whichstrand
whichstrand(GPP) <- "+"

whichstrand<-methods

Description

Setter method for the whichstrand slot in a parameterOptions object

Methods:

whichstrand(object)<-value
Index

* classes
  ChIPScore-class, 20
  genomicProfiles-class, 43
  genomicProfilesInternal-class, 46
  GRList-class, 50
  loci-class, 54
  nos-class, 71
  parameterOptions-class, 74

* methods
  averageExpPWMScore-methods, 7
  backgroundSignal-methods, 9
  backgroundSignal<-=methods, 10
  boundMolecules-methods, 12
  boundMolecules<-=methods, 13
  BPFrequency-methods, 14
  BPFrequency<-=methods, 16
  chipMean-methods, 18
  chipMean<-=methods, 20
  chipSd-methods, 23
  chipSd<-=methods, 24
  chipSmooth-methods, 25
  chipSmooth<-=methods, 26
  DNASequenceLength-methods, 37
  drop-methods, 39
  initialize-methods, 51
  lambdaPWM-methods, 52
  lambdaPWM<-=methods, 53
  loci-methods, 55
  lociWidth-methods, 56
  lociWidth<-=methods, 58
  maxPWMScore-methods, 59
  maxSignal-methods, 60
  maxSignal<-=methods, 62
  minPWMScore-methods, 63
  naturalLog-methods, 64
  naturalLog<-=methods, 66
  noiseFilter-methods, 67
  noiseFilter<-=methods, 68
  noOfSites<-=methods, 71
  PFMFormat-methods, 78
  PFMFormat<-=methods, 79
  ploidy-methods, 80
  ploidy<-=methods, 82
  PositionFrequencyMatrix-methods, 87
  PositionFrequencyMatrix<-=methods, 88
  PositionWeightMatrix-methods, 90
  PositionWeightMatrix<-=methods, 91
  profiles-methods, 95
  PWMpseudocount-methods, 96
  PWMpseudocount<-=methods, 97
  PWMThreshold-methods, 99
  PWMThreshold<-=methods, 100
  removeBackground-methods, 101
  removeBackground<-=methods, 102
  scores-methods, 104
  show-methods, 107
  stepSize-methods, 110
  stepSize<-=methods, 111
  strandRule-methods, 112
  strandRule<-=methods, 114
  whichstrand-methods, 115
  whichstrand<-=methods, 116

* package
  ChIPanalyser-package, 5
  .DNASequenceLength<-,genomicProfilesInternal,vector-method
    (genomicProfilesInternal-class), 46
  .ZeroBackground,parameterOptions-method
    (parameterOptions-class), 74
  .ZeroBackground<-,parameterOptions,vector-method
    (parameterOptions-class), 74
  .averageExpPWMScore<-,genomicProfilesInternal,numeric-method
    (genomicProfilesInternal-class), 46
  .drop<-,genomicProfilesInternal,vector-method
INDEX

noOfSites, parameterOptions-method
    (parameterOptions-class), 74
noOfSites-methods, 69
noOfSites<-, 70
noOfSites<-methods, 71
noOfSites<-, parameterOptions, character-method
    (parameterOptions-class), 74
noOfSites<-, parameterOptions, numeric-method
    (parameterOptions-class), 74
nos-class, 71
parameterOptions, 8–12, 17–19, 21–28, 31,
    33, 35, 43, 45, 47, 51–53, 55–57,
    59–61, 63–70, 72, 79–82, 92, 93,
    95–102, 109–116
parameterOptions-class, 74
PFMFormat, 77
PFMFormat, genomicProfilesInternal-method
    (genomicProfilesInternal-class), 46
PFMFormat-methods, 78
PFMFormat<-, 78
PFMFormat<-methods, 79
PFMFormat<-, genomicProfilesInternal, character-method
    (genomicProfilesInternal-class), 46
ploidy, 72, 79
ploidy, parameterOptions-method
    (parameterOptions-class), 74
ploidy-methods, 80
ploidy<-, 81
ploidy<-methods, 82
ploidy<-, parameterOptions, numeric-method
    (parameterOptions-class), 74
plotOccupancyProfile, 8, 9, 82
plotOptimalHeatMaps, 84
PositionFrequencyMatrix, 77, 78, 86
PositionFrequencyMatrix, genomicProfilesInternal-method
    (genomicProfilesInternal-class), 46
PositionFrequencyMatrix-methods, 87
PositionFrequencyMatrix<-, 87
PositionFrequencyMatrix<-, methods, 88
PositionFrequencyMatrix<-, genomicProfilesInternal, character-method
    (genomicProfilesInternal-class), 46
PositionFrequencyMatrix<-, genomicProfilesInternal-method
    (genomicProfilesInternal-class), 46
PositionWeightMatrix, 86, 89
PositionWeightMatrix, genomicProfilesInternal-method
    (genomicProfilesInternal-class), 46
PositionWeightMatrix-methods, 90
PositionWeightMatrix<-, 90
PositionWeightMatrix<-methods, 91
PositionWeightMatrix<-, genomicProfilesInternal, matrix-method
    (genomicProfilesInternal-class), 46
profiles-methods, 95
PWMpseudocount, 73, 95
PWMpseudocount, parameterOptions-method
    (parameterOptions-class), 74
PWMpseudocount-methods, 96
PWMpseudocount<-, 96
PWMpseudocount<-methods, 97
PWMpseudocount<-, parameterOptions, numeric-method
    (parameterOptions-class), 74
PWMThreshold, 29, 34, 35, 73, 98
PWMThreshold, parameterOptions-method
    (parameterOptions-class), 74
PWMThreshold-methods, 99
PWMThreshold<-, 99
PWMThreshold<-methods, 100
PWMThreshold<-, parameterOptions, numeric-method
    (parameterOptions-class), 74
removeBackground, 72, 100
removeBackground, parameterOptions-method
    (parameterOptions-class), 74
removeBackground-methods, 101
removeBackground<-, 101
removeBackground<-methods, 102
removeBackground<-, parameterOptions, vector-method
    (parameterOptions-class), 74
scores, 103
scores, ChIPScore-method
    (ChIPScore-class), 20
scores-methods, 104
INDEX

searchSites, 82, 104
setChromatinStates, 106
show, ChIPScore-method (show-methods), 107
show, genomicProfiles-method
  (show-methods), 107
show, parameterOptions-method
  (show-methods), 107
show-methods, 107
singleRun, 107
splitData, 108
stepSize, 72, 109
stepSize, parameterOptions-method
  (parameterOptions-class), 74
stepSize-methods, 110
stepSize<-, 110
stepSize<-, methods, 111
stepSize<-, parameterOptions, numeric-method
  (parameterOptions-class), 74
strandRule, 73, 111
strandRule, parameterOptions-method
  (parameterOptions-class), 74
strandRule-methods, 112
strandRule<-, 113
strandRule<-, methods, 114
strandRule<-, parameterOptions, character-method
  (parameterOptions-class), 74

top (ChIPanalyserData), 16

whichstrand, 73, 112, 113, 114
whichstrand, parameterOptions-method
  (parameterOptions-class), 74
whichstrand-methods, 115
whichstrand<-, 115
whichstrand<-, methods, 116
whichstrand<-, parameterOptions, character-method
  (parameterOptions-class), 74