

Package ‘nucleR’

October 9, 2015

Type Package

Title Nucleosome positioning package for R

Version 2.0.0

Date 2015-02-20

Author Oscar Flores, Ricard Illa

Maintainer Ricard Illa <ricard.illa@irbbarcelona.org>

Description Nucleosome positioning for Tiling Arrays and NGS experiments.

License LGPL (>= 3)

Depends ShortRead

Imports methods, BiocGenerics, IRanges, Biobase, GenomicRanges,
Rsamtools, stats, graphics, parallel, S4Vectors

Suggests Starr

LazyLoad yes

biocViews ChIPSeq, Microarray, Sequencing, Genetics

NeedsCompilation no

R topics documented:

nucleR-package	2
controlCorrection	3
coverage.rpm	5
export.bed	6
export.wig	7
filterFFT	8
fragmentLenDetect	10
mergeCalls	12
nucleosome_htseq	13
nucleosome_tiling	14
pcKeepCompDetect	15
peakDetection	16
peakScoring	18

plotPeaks	20
processReads	21
processTilingArray	23
readBAM	26
syntheticNucMap	26

Index	29
--------------	-----------

nucleR-package	<i>Nucleosome positioning package for R</i>
----------------	---

Description

Nucleosome positioning from Tiling Arrays and High-Troughput Sequencing Experiments

Details

```

Package:  nucleR
Type:    Package
License:  LGPL (>= 3)
LazyLoad: yes

```

This package provides a convenient pipeline to process and analyze nucleosome positioning experiments from High-Troughput Sequencing or Tiling Arrays.

Despite it's use is intended to nucleosome experiments, it can be also useful for general ChIP experiments, such as ChIP-on-ChIP or ChIP-Seq.

See following example for a brief introduction to the available functions

Author(s)

Oscar Flores Ricard Illa

Maintainer: Ricard Illa <ricard.illa@irbbarcelona.org>

Examples

```

#Load example dataset:
# some NGS paired-end reads, mapped with Bowtie and processed with R
# it is a RangedData object with the start/end coordinates for each read.
reads = get(data(nucleosome_htseq))

#Process the paired end reads, but discard those with length > 200
preads_orig = processReads(reads, type="paired", fragmentLen=200)

#Process the reads, but now trim each read to 40bp around the dyad
preads_trim = processReads(reads, type="paired", fragmentLen=200, trim=40)

```

```

#Calculate the coverage, directly in reads per million (r.p.m)
cover_orig = coverage.rpm(preads_orig)
cover_trim = coverage.rpm(preads_trim)

#Compare both coverages, the dyad is much more clear in trimmed version
t1 = as.vector(cover_orig[[1]][1:2000])
t2 = as.vector(cover_trim[[1]][1:2000])
t1 = (t1-min(t1))/max(t1-min(t1)) #Normalization
t2 = (t2-min(t2))/max(t2-min(t2)) #Normalization
plot(t1, type="l", lwd="2", col="blue", main="Original vs Trimmed coverage")
lines(t2, lwd="2", col="red")
legend("bottomright", c("Original coverage", "Trimmed coverage"), lwd=2, col=c("blue", "red"), bty="n")

#Let's try to call nucleosomes from the trimmed version
#First of all, let's remove some noise with FFT
#Power spectrum will be plotted, look how with a 2%
#of the components we capture almost all the signal
cover_clean = filterFFT(cover_trim, pcKeepComp=0.02, showPowerSpec=TRUE)

#How clean is now?
plot(as.vector(cover_trim[[1]][1:4000]), t="l", lwd=2, col="red", main="Noisy vs Filtered coverage")
lines(cover_clean[[1]][1:4000], lwd=2, col="darkgreen")
legend("bottomright", c("Input coverage", "Filtered coverage"), lwd=2, col=c("red", "darkgreen"), bty="n")

#And how similar? Let's see the correlation
cor(cover_clean[[1]], as.vector(cover_trim[[1]]))

#Now it's time to call for peaks, first just as points
#See that the score is only a measure of the height of the peak
peaks = peakDetection(cover_clean, threshold="25%", score=TRUE)
plotPeaks(peaks[[1]], cover_clean[[1]], threshold="25%")

#Do the same as previously, but now we will create the nucleosome calls:
peaks = peakDetection(cover_clean, width=147, threshold="25%", score=TRUE)
plotPeaks(peaks, cover_clean[[1]], threshold="25%")

#This is all. From here, you can filter, merge or work with the nucleosome
#calls using standard IRanges functions and R/Bioconductor manipulation

```

controlCorrection

Correct experimental profiles with control sample

Description

This function allows the correction of experimental coverage profiles (usually MNase digested nucleosomal DNAs in this library) with control samples (usually naked DNA sample digested with MNase). This is useful to correct MNase biase.

Usage

```
## S4 method for signature 'SimpleRleList'  
controlCorrection(exp, ctr, mc.cores=1)  
## S4 method for signature 'Rle'  
controlCorrection(exp, ctr)  
## S4 method for signature 'list'  
controlCorrection(exp, ctr, mc.cores=1)  
## S4 method for signature 'numeric'  
controlCorrection(exp, ctr)
```

Arguments

exp, ctr	Comparable experimental and control samples (this means same format and equivalent preprocessing)
mc.cores	Number of cores available for parallel list processing

Details

This subtracts the enrichment in the control sample respect it's mean from the experimental profile.

This is useful for examining the effect of the MNase digestion in nucleosome experiments using a nucleosomal DNA and a genomic (naked) DNA sample. Notice that genomic DNA samples cannot be strand-corrected using single end data, so only paired end controls are useful for this proupose, despite they can be compared against extended nucleosomal DNA single end reads. Furthermore, both datasets must be converted to reads per milion.

This process difcults the nucleosome positioning due the lower sharpness of the peaks, but allows a complementary study of the MNase digestion effect.

Value

Corrected experimental profile

Author(s)

Oscar Flores <oflores@mmb.pcb.ub.es>

Examples

```
#Toy example  
map = syntheticNucMap(as.ratio=TRUE)  
  
exp = coverage(map$syn.reads)  
ctr = coverage(map$ctr.reads)  
  
corrected = controlCorrection(exp, ctr)
```

coverage.rpm	<i>Coverage calculation and normalization to reads per million (rpm)</i>
--------------	--

Description

Calculates the coverage values from a RangedData object (or anything with a defined coverage function associated) and returns the coverage normalized to reads per million, allowing the comparison of experiments with a different absolute number of reads.

Usage

```
coverage.rpm(data, scale=1e6, ...)
```

Arguments

data	RangedData (or compatible) with the reads information
scale	By default, a million (1e6), but you could change this value for abnormal high or low amount of reads
...	Additional arguments to be passed to coverage function

Value

RleList object with the coverage objects

Author(s)

Oscar Flores <oflores@mmb.pcb.ub.es>

See Also

[processReads](#), [coverage](#)

Examples

```
#Load the example dataset and get the coverage
data(nucleosome_htseq)
cov = coverage.rpm(nucleosome_htseq)

print(cov)

#Plot it
plot(as.vector(cov[["chr1"]]), type="l", ylab="coverage", xlab="position")
```

 export.bed

Export ranges in BED format

Description

Export ranges in BED format, compatible with UCSC genome browser, IGB, and others

Usage

```

    ## S4 method for signature 'IRanges'
  export.bed(ranges, score=NULL, chrom, name, desc=name, filepath=name)
    ## S4 method for signature 'CompressedIRangesList'
  export.bed(ranges, score=NULL, name, desc=name, filepath=name, splitByChrom=TRUE)
    ## S4 method for signature 'RangedData'
  export.bed(ranges, score=NULL, name, desc=name, filepath=name, splitByChrom=TRUE)
    ## S4 method for signature 'GRanges'
  export.bed(ranges, score=NULL, name, desc=name, filepath=name, splitByChrom=TRUE)

```

Arguments

ranges	Ranges to export, in IRanges, IRangesList or RangedData format
score	Score data if not included in ranges object. Bed file will put all scores=1000 if scores are not present
chrom	For single IRanges objects, the chromosome they represent. For other data types, values from names(...) will be used.
name	Name of the track
desc	Description of the track
filepath	Path and prefix of the file(s) to write. Chromosome number and "bed" extension will be automatically added.
splitByChrom	If multiple chromosomes are given, should they be splitted into one file per chromosome or shall them be saved all together?

Value

(none)

Author(s)

Oscar Flores <oflores@mmb.pcb.ub.es>

References

BED format specification: <http://genome.ucsc.edu/FAQ/FAQformat#format1>

Examples

```
## Not run:
# Generate random ranges with scores
ran <- RangedData(IRanges(start=1:100, end=101:200), score=(1:100) / 100)
names(ran) <- "chrX"

# Export as bed file
export.bed(ran, name="test_track", description="Just a test track")

# If executed, this would create a file named "test_track.chrX.bed" with:

#   track name="test_track" description="Just a test track" useScore=0
# chrX   1 101 nucl1 0.01
# chrX   2 102 nucl2 0.02
# chrX   3 103 nucl3 0.03
# ...

## End(Not run)
```

export.wig

Export values in WIG format

Description

Export coverage/intensity values in WIG format, compatible with UCSC genome browser, IGB, and others

Usage

```
export.wig(data, name, chrom="", filepath=name)
```

Arguments

data	Coverage/intensity values (numeric)
name	Name of the track
chrom	Information about chromosome if not inferrable from data (only for numeric vectors)
filepath	Filepath where to save the object. Chromosome name and "wig" extension will be automatically added

Value

(none)

Author(s)

Oscar Flores <oflores@mmb.pcb.ub.es>

References

WIG format specification: <http://genome.ucsc.edu/FAQ/FAQformat#format6>

Examples

```
## Not run:
#Load data
data(nucleosome_htseq)
cover = coverage.rpm(nucleosome_htseq)

#Create wig file
export.wig(cover, name="example_track")

#This would create the file "example_track.chr1.wig" with:

#track type=wiggle_0 name="example_track"
#fixedStep chrom=chr1 start=1 step=1
#55.55247
#55.55247
#55.55247
#277.7623
#388.8673
#...

## End(Not run)
```

filterFFT

Clean noise and smoothing for genomic data using Fourier-analysis

Description

Remove noise from genomic data smoothing and cleaning the observed signal. This function doesn't alter the shape or the values of the signal as much as the traditional method of sliding window average does, providing a great correlation within the original and filtered data (>0.99).

Usage

```
## S4 method for signature 'SimpleRleList'
filterFFT(data, pcKeepComp="auto", showPowerSpec=FALSE, useOptim=TRUE, mc.cores=1, ...)
## S4 method for signature 'list'
filterFFT(data, pcKeepComp="auto", showPowerSpec=FALSE, useOptim=TRUE, mc.cores=1, ...)
## S4 method for signature 'Rle'
filterFFT(data, pcKeepComp="auto", showPowerSpec=FALSE, useOptim=TRUE, ...)
## S4 method for signature 'numeric'
filterFFT(data, pcKeepComp="auto", showPowerSpec=FALSE, useOptim=TRUE, ...)
```


Arguments

data	Coverage or intensities values representing the results of the NGS of TA experiment. This attribute could be a individual vector representing a chromosome (R1e or numeric object) or a list of them.
pcKeepComp	Number of components to select, in percentage respect total length of the sample. Allowed values are numeric (in range 0:1) for manual setting or "auto" for automatic detection. See details.
showPowerSpec	Plot the Power Spectrum of the Fast Fourier Transform to visually identify the selected components (see details).
useOptim	This function implements tweaks to a standard fft call to improve (dramatically) the performance in large genomic data. These optimizations can be bypassed by setting this parameter to FALSE.
mc.cores	If multiple cores are available, maximum number of them to use for parallel processing of data elements (only useful if data is a list of elements)
...	Other parameters to be passed to pcKeepCompDetect function

Details

Fourier-analysis principal components selection is widely used in signal processing theory for an unbiased cleaning of a signal over the time.

Other procedures, as the traditional sliding window average, can change too much the shape of the results in function of the size of the window, and moreover they don't only smooth the noise without removing it.

With a Fourier Transform of the original signal, the input signal is decomposed in diferent wavelets and described as a combination of them. Long frequencies can be explained as a function of two ore more periodical shorter frecuencies. This is the reason why long, unperiodic sequences are usually identified as noise, and therefore is desireable to remove them from the signal we have to process.

This procedure here is applied to genomic data, providing a novel method to obtain perfectly clean values wich allow an efficient detection of the peaks which can be used for a direct nucleosome position recognition.

This function select a certain number of components in the original power spectrum (the result of the Fast Fourier Transform which can be seen with showPowerSpec=TRUE) and sets the rest of them to 0 (component knock-out).

The amout of components to keep (given as a percentage of the input lenght) can be set by the pcKeepComp. This will select the first components of the signal, knock-outing the rest. If this value is close to 1, more components will be selected and then more noise will be allowed in the output. For an effective filtering which removes the noise keeping almost all relevant peaks, a value between 0.01 and 0.05 is usually sufficient. Lower values can cause merging of adjacent minor peaks.

This library also allows the automatic detection of a fitted value for pcKeepComp. By default, it uses the pcKeepCompDetect function, which looks which is the minimum percentage of components than can reproduce the original signal with a corelation between the filtered and the original one of 0.99. See the help page of pcKeepCompDetect for further details and reference of available parameters.

One of the most powerful features of nucleR is the efficient implementation of the FFT to genomic data. This is achieved through few tweaks that allow an optimum performance of the Fourier Transform. This includes a by-range filtering, an automatic detection of uncovered regions, windowed execution of the filter and padding of the data till nearest power of 2 (this ensures an optimum case for FFT due to the high factorization of components). Internal testing showed up that in specific datasets, these optimizations lead to a dramatic improvement of many orders of magnitude (from 3 days to few seconds) while keeping the correlation between the native `fft` call and our `filterFFT` higher than 0.99. So, the use of these optimizations is highly recommended.

If for some reason you want to apply the function without any kind of optimizations you can specify the parameter `useOptim=FALSE` to bypass them and get the pure knockout inverse from native FFT call. All other parameters can be still applied in this case.

Value

Numeric vector with cleaned/smoothed values

Author(s)

Oscar Flores <oflores@mmb.pcb.ub.es>, David Rosell <david.rossell@irbbarcelona.org>

References

Smith, Steven W. (1999), *The Scientist and Engineer's Guide to Digital Signal Processing* (Second ed.), San Diego, Calif.: California Technical Publishing, ISBN 0-9660176-3-3 (available online: <http://www.dspguide.com/pdfbook.htm>)

Examples

```
#Load example data, raw hybridization values for Tiling Array
raw_data = get(data(nucleosome_tiling))

#Filter data
fft_data = filterFFT(raw_data, pcKeepComp=0.01)

#See both profiles
par(mfrow=c(2,1), mar=c(3, 4, 1, 1))
plot(raw_data, type="l", xlab="position", ylab="Raw intensities")
plot(fft_data, type="l", xlab="position", ylab="Filtered intensities")

#The power spectrum shows a visual representation of the components
fft_data = filterFFT(raw_data, pcKeepComp=0.01, showPowerSpec=TRUE)
```

Description

When using single-ended sequencing, the resulting partial sequences map only in one strand, causing a bias in the coverage profile if not corrected. The only way to correct this is knowing the average size of the real fragments. `nucleR` uses this information when preprocessing single-ended sequences. You can provide this information by your own (usually a 147bp length is a good approximation) or you can use this method to automatically guess the size of the inserts.

Usage

```
## S4 method for signature 'AlignedRead'
fragmentLenDetect(reads, samples=1000, window=1000, min.shift=1, max.shift=100, mc.cores=1, as.shift=TRUE)
## S4 method for signature 'RangedData'
fragmentLenDetect(reads, samples=1000, window=1000, min.shift=1, max.shift=100, mc.cores=1, as.shift=TRUE)
```

Arguments

<code>reads</code>	Raw single-end reads (AlignedRead or RangedData format)
<code>samples</code>	Number of samples to perform the analysis (more = slower but more accurate)
<code>window</code>	Analysis window. Usually there's no need to touch this parameter.
<code>min.shift</code> , <code>max.shift</code>	Minimum and maximum shift to apply on the strands to detect the optimal fragment size. If the range is too big, the performance decreases.
<code>as.shift</code>	If TRUE, returns the shift needed to align the middle of the reads in opposite strand. If FALSE, returns the mean inferred fragment length.
<code>mc.cores</code>	If multicore support, maximum number of cores allowed to use.

Details

This function shifts one strand downstream one base by one from `min.shift` to `max.shift`. In every step, the correlation on a random position of length `window` is checked between both strands. The maximum correlation is returned and averaged for `samples` repetitions.

The final returned length is the best shift detected plus the width of the reads. You can increase the performance of this function by reducing the `samples` value and/or narrowing the shift range. The window size has almost no impact on the performance, despite a too small value can give biased results.

Value

Inferred mean length of the inserts by default, or shift needed to align strands if `as.shift=TRUE`

Author(s)

Oscar Flores <oflores@mmb.pcb.ub.es>

Examples

```
#Create a synthetic dataset, simulating single-end reads, for positive and negative strands
pos = syntheticNucMap(nuc.len=40, lin.len=130)$syn.reads #Positive strand reads
neg = IRanges(end=start(pos)+147, width=40) #Negative strand (shifted 147bp)
sim = RangedData(c(pos, neg), strand=c(rep("+", length(pos)), rep("-", length(neg))))

#Detect fragment length (we know by construction it is really 147)
fragmentLenDetect(sim, samples=50)

#The function restrict the sampling to speed up the example
```

mergeCalls

Automatic merging of overlapped nucleosome calls

Description

This function joints close nucleosome calls into one larger, fuzzy nucleosome.

Usage

```
mergeCalls(calls, min.overlap=50, discard.low=0.2, mc.cores=1, verbose=TRUE)
```

Arguments

calls	RangedData with scored and ranged nucleosome calls from peakScoring or peakDetection(..., score=TRUE).
min.overlap	Minimum overlap between two reads for merge them
discard.low	Discard low covered calls (i.e. calls with score_h < discard.low will be discarded)
mc.cores	Number of cores available to parallel data processing.
verbose	Show progress info?

Details

This functions looks for overlapped calls and join those with more than min.overlap bases overlapped. More than two reads can be joined in one single call if all of them are overlapped at least that distance with almost another read in the range.

Joining is performed in chain, so if nucleosome call A is close to B and B is close to C, the final call will comprise the range A-B-C. The resulting scores (mixed, width, height) of the final joined call will be the average value of the individual scores.

The parameter discard.low allows to ignore the small peaks that could be merged with larger ones, originating large calls. In the case that all of the overlapped reads in a given position have score_h less than discard.low, all of them will be selected instead of deleting that call.

Value

RangedData with merged calls and the additional data column nmerge, with the count of how many original ranges are merged in the resulting range.

Author(s)

Oscar Flores <oflores@mmb.pcb.ub.es>

See Also

[peakScoring](#)

Examples

```
#Generate a synthetic coverage map (assuming reads of 40bp and fragments of 130)
map = syntheticNucMap(wp.num=20, fuz.num=20, nuc.len=40, lin.len=130, rnd.seed=1)
cover = filterFFT(coverage(map$syn.reads))

#Find peaks over FFT filtered coverage
calls = peakDetection(filterFFT(cover, pcKeepComp=0.02), width=130, score=TRUE)

#Merge overlapped calls
merged_calls = mergeCalls(calls)

plotPeaks(merged_calls, cover)
```

nucleosome_htseq	<i>Example reads from high-throughput sequencing nucleosome positioning experiment</i>
------------------	--

Description

Few reads from paired-ended MNase-seq experiment in *S.cerevisiae* where mononucleosomes were sequenced

Usage

```
data(nucleosome_htseq)
```

Format

RangedData with the range of the reads and a data column with the strand information.

Details

This data is obtained from MNase digested nucleosomal DNA and sequenced with Illumina platform. Paired-ended reads were mapped to SacCer1 genome using Bowtie, and imported to R using the package ShortRead and paired ends were merged into a single range.

Reads were sorted by chromosome and starting position and only a few reads from the starting positions of chromosome 1 are presented.

Source

Publication pending

nucleosome_tiling	<i>Example intensities from Tiling Microarray nucleosome positioning experiment</i>
-------------------	---

Description

Some bases from *S.cerevisiae* tiling microarray where mononucleosomes were sequenced and hybridized with histone-free naked DNA. The intensity is the normalized ratio between the intensities from nucleosomic and naked DNA.

Due to the difficulty of providing a raw file, this file has been preprocessed. See details.

Usage

```
data(nucleosome_tiling)
```

Format

numeric vector with the intensities.

Details

The raw .CEL files from Affymetrix *S.Cerevisiae* Tiling 1.0R Array (3 nucleosomal + 3 naked DNA) has been merged using package *Starr* and the resulting *ExpressionSet* object has been passed to *processTilingArray* function from this package as follows:

```
processTilingArray(data, exprName, chrPattern="Sc:Oct_2003;chr1", closeGaps=50)
```

The first 8000bp of the chr1 have been saved as this example dataset.

Source

Publication pending

pcKeepCompDetect *Auto detection of a fitted pcKeepComp param for filterFFT function*

Description

This function tries to obtain the minimum number of components needed in a FFT filter to achieve or get as close as possible to a given correlation value. Usually you don't need to call directly this function, is used in filterFFT by default.

Usage

```
pcKeepCompDetect(data, pc.min=0.01, pc.max=0.1, max.iter=20, verbose=FALSE,
  cor.target=0.98, cor.tol=1e-3, smp1.num=25, smp1.min.size=2^10, smp1.max.size=2^14)
```

Arguments

data	Numeric vector to be filtered
pc.min, pc.max	Range of allowed values for pcKeepComp (minimum and maximum), in the range 0:1.
max.iter	Maximum number of iterations
verbose	Extra information (debug)
cor.target	Target correlation between the filtered and the original profiles. A value around 0.99 is recommended for Next Generation Sequencing data and around 0.7 for Tiling Arrays.
cor.tol	Tolerance allowed between the obtained correlation and the target one.
smp1.num	If data is a large vector, some samples from the vector will be used instead the whole dataset. This parameters tells the number of samples to pick.
smp1.min.size, smp1.max.size	Minimum and maximum size of the samples. This is used for selection and sub-selection of ranges with meaningful values (i.e, different from 0 and NA). Power of 2 values are recommended, despite non-mandatory.
...	Parameters to be pass to autoPcKeepComp

Details

This function predicts a suitable pcKeepComp value for filterFFT function. This is the recommended amount of components (in percentage) to keep in the filterFFT function to obtain a correlation of (or near of) cor.target.

The search starts from two given values pc.min, pc.max and uses linial interpolation to quickly reach a value that gives a corelation between the filtered and the original near cor.target within the specified tolerance cor.tol.

To allow a quick detection without an exhaustive search, this function uses a subset of the data by randomly sampling those regions with meaningful coverage values (i.e, different from 0 or NA) larger than smp1.min.size. If it's not possible to obtain smp1.max.size from this region (this

could be due to flanking 0's, for example) at least `smp1.min.size` will be used to check correlation. Mean correlation between all sampled regions is used to test the performance of the `pcKeepComp` parameter.

If the number of meaningful bases in data is less than `smp1.min.size * (smp1.num/2)` all the data vector will be used instead of using sampling.

Value

Fitted `pcKeepComp` value

Author(s)

Oscar Flores <oflores@mmb.pcb.ub.es>, David Rosell <david.rosell@irbbarcelona.org>

Examples

```
#Load dataset
data(nucleosome_htseq)
data = as.vector(coverage.rpm(nucleosome_htseq)[[1]])

#Get recommended pcKeepComp value
pkeepcomp = pcKeepCompDetect(data, cor.target=0.99)
print(pkeepcomp)

#call filterFFT
f1 = filterFFT(data, pcKeepComp=pkeepcomp)

#Also this can be called directly
f2 = filterFFT(data, pcKeepComp="auto", cor.target=0.99)

#Plot
plot(data[1:2000], col="black", type="l", lwd=2)
lines(f1[1:2000], col="red", lwd=2)
lines(f2[1:2000], col="blue", lwd=2, lty=2)
legend("bottom", c("original", "two calls", "one call"), col=c("black", "red", "blue"), lty=c(1,1,2), horiz=TRUE)
```

peakDetection

Detect peaks (local maximum) from values series

Description

This function allows a efficient recognition of the local maximums (peaks) in a given numeric vector.

It's recommended to smooth the input with `filterFFT` prior the detection.

Usage

```
## S4 method for signature 'list'
peakDetection(data, threshold="25%", width=1, score=TRUE, mc.cores=1)
## S4 method for signature 'numeric'
peakDetection(data, threshold="25%", width=1, score=TRUE, mc.cores=1)
```

Arguments

data	Input numeric values, or a list of them
threshold	Threshold value from which the peaks will be selected. Can be given as a percentage string (i.e., "25%" will use the value in the 1st quantile of data) or as an absolute coverage numeric value (i.e., 20 will not look for peaks in regions without less than 20 reads (or reads per milion)).
width	If a positive integer > 1 is given, the peaks are returned as a range of the given width centered in the local maximum. Useful for nucleosome calling from a coverage peak in the dyad.
score	If TRUE, the results will be scored using peakScoring function
mc.cores	If multicore support, the number of cores available

Value

The type of the return depends on the input parameters:

numeric (or a list of them) if width==1 & score==FALSE containing the position of the peaks

data.frame (or list of them) if width==1 & score==TRUE containing a 'peak' column with the position of the peak plus a 'score' column with its score.

IRanges (or IRangesList) if width>1 & score==FALSE containing the ranges of the peaks.

RangedData if width>1 & score==TRUE containing the ranges of the peaks and the assigned score.

Note

If width > 1, those ranges outside the range 1:length(data) will be skipped

Author(s)

Oscar Flores <oflores@mmb.pcb.ub.es>

See Also

[filterFFT](#), [peakScoring](#)

Examples

```
#Generate a random peaks profile
reads = syntheticNucMap(nuc.len=40, lin.len=130)$syn.reads
cover = coverage(reads)
```

```
#Filter them
cover_fft = filterFFT(cover)

#Detect and plot peaks (up a bit the threshold for accounting synthetic data)
peaks = peakDetection(cover_fft, threshold="40%", score=TRUE)
plotPeaks(peaks, cover_fft, threshold="40%", start=10000, end=15000)

#Now use ranges version, which accounts for fuzziness when scoring
peaks = peakDetection(cover_fft, threshold="40%", score=TRUE, width=147)
plotPeaks(peaks, cover_fft, threshold="40%", start=10000, end=15000)
```

peakScoring	<i>Peak scoring function</i>
-------------	------------------------------

Description

Scores peaks detected with function `peakDetection` according the height and the sharpness (width) of the peak. This function can be called automatically from `peakDetection` if `score=TRUE`.

Usage

```
## S4 method for signature 'numeric'
peakScoring(peaks, data, threshold="25%")
## S4 method for signature 'list'
peakScoring(peaks, data, threshold="25%", mc.cores=1)
## S4 method for signature 'IRanges'
peakScoring(peaks, data, threshold="25%", weight.width=1, weight.height=1, dyad.length=38)
## S4 method for signature 'IRangesList'
peakScoring(peaks, data, threshold="25%", weight.width=1, weight.height=1, dyad.length=38, mc.cores=1)
```

Arguments

peaks	The identified peaks resulting from <code>peakDetection</code> . Could be a numeric vector with the position of the peaks, or a <code>IRanges</code> object with the extended range of the peak. For both types, list support is implemented as a numeric list or a <code>IRangesList</code>
data	Data of nucleosome coverage or intensities.
threshold	The non-default threshold previously used in <code>peakDetection</code> function, if applicable. Can be given as a percentage string (i.e., "25%" will use the value in the 1st quantile of data) or as an absolute coverage numeric value (i.e., 20 will not look for peaks in regions without less than 20 reads (or reads per milion)).
dyad.length	How many bases account in the nucleosome dyad for sharpness description. If working with NGS data, works best with the reads width value for single-ended data or the trim value given to the <code>processReads</code> function.
weight.height, weight.width	If the score is a range, the height and the width score (coverage and fuzzyness) can be defined with different weights with these parameters. See details.

`mc.cores` If input is a list or `IRangeList`, and multiple cores support is available, the maximum number of cores for parallel processing

Details

This function scores each previously identified peak according its height and sharpness.

The height score (`score_h`) tells how large is a peak, higher means more coverage or intensity, so better positioned nucleosome. This score is obtained by checking the observed peak value in a Normal distribution with the mean and sd of data. This value is between 0 and 1.

The width score (`score_w`) is a mesure of how sharp is a peak. With a NGS coverage in mind, a perfect phased (well-positioned) nucleosome is this that starts and ends exactly in the same place many times. The shape of this ideal peak will be a rectangular shape of the lenght of the read. A wider top of a peak could indicate fuzzyness. The parameter `dyad.length` tells how long should be the "flat" region of an ideal peak. The optimum value for this parameter is the lenght of the read in single-ended data or the `trim` value of the function `processReads`. For Tiling Array, the default value should be `fine`. This score is obtained calculating the ratio between the mean of the nucleosome scope (the one provided by range in the elements of peaks) and the `dyad.length` central bases. This value is normalized between 0 and 1.

For punctual, single points peaks (provided by numeric vector or list as peaks attribute) the score returned is the height score.

For range peaks the weighted sum of the heigth and width scores is used. This is: $((score_h * weighth.height) / sum.weigh)$
Note that you can query for only one score by weting its weight to 1 and the other to 0.

Value

In the case of numeric input, the value returned is a `data.frame` containing a 'peak' and a 'score' column. If the input is a list, the result will be a list of `data.frame`.

If input is a `IRanges` or `IRangesList`, the result will be a `RangedData` object with one or multiple spaces respectively and a 3 data column with the mixed, width and heigh score.

Author(s)

Oscar Flores <oflores@mmb.cpb.ub.es>

See Also

[peakDetection](#), [processReads](#),

Examples

```
#Generate a synthetic map
map = syntheticNucMap(nuc.len=40, lin.len=130) #Trimmed length nucleosome map

#Get the information of dyads and the coverage
peaks = c(map$wp.starts, map$fz.starts)
cover = filterFFT(coverage(map$syn.reads))

#Calculate the scores
```

```
scores = peakScoring(peaks, cover)
plotPeaks(scores$peak, cover, scores=scores$score, start=5000, end=10000)
```

plotPeaks

Nucleosome calling plot function

Description

Helper function for a quick and convenient overview of nucleosome calling data.

This function is intended to plot data previously processed with nucleR pipeline. It shows a coverage/intensity profile together with the identified peaks. If available, score of each peak is also shown.

Usage

```
plotPeaks(peaks, data, ...)
## S4 method for signature 'IRanges'
plotPeaks(peaks, data, threshold=0, scores=NULL, start=1,
end=length(data), dyn.pos=TRUE, xlab="position", type="l", col.points="red",
thr.lty=1, thr.lwd="1", thr.col="darkred", rect.thick=2, rect.lwd=1,
rect.border="black", scor.col=col.points, scor.font=2, scor.adj=c(0.5,0),
scor.cex=0.75, scor.digits=2, indiv.scores=TRUE, ...)
## S4 method for signature 'numeric'
plotPeaks(peaks, data, threshold=0, scores=NULL, start=1,
end=length(data), xlab="position", type="l", col.points="red",
thr.lty=1, thr.lwd="1", thr.col="darkred", scor.col=col.points, scor.font=2,
scor.adj=c(0.5,0), scor.cex=0.75, scor.digits=2,...)
```

Arguments

peaks	numeric, data.frame, IRanges or RangedData object containing the detected peaks information. See help of peakDetection or peakScoring for more details.
data	Coverage or Tiling Array intensities
threshold	Threshold applied in peakDetection
scores	If peaks is a data.frame or a RangedData it's obtained from 'score' column, otherwise, scores can be given here as a numeric vector
start, end	Start and end points defining a subset in the range of data. This is a convenient way to plot only a small region of data, without dealing with subsetting of range or score objects.
dyn.pos	If peaks are ranges, should they be positioned dynamically on top of the peaks or statically at threshold baseline. Spacing of overlapping ranges is automatically applied if FALSE.
xlab, type, col.points	Default values to be passed to plot and points

thr.lty, thr.lwd, thr.col
 Default values to be passed to abline for threshold representation

rect.thick, rect.lwd, rect.border
 Default values for rect representation or ranges. rect.thick indicates the thickness (in percentage relative to y-axis range) of the rectangles.

scor.col, scor.font, scor.adj, scor.cex, scor.digits
 Default values for text representation for score numbers, if available.

indiv.scores Show or hide individual scores for width and height in brackets besides the mixed score

... Other parameters passed to [plot](#) function

Value

(none)

Author(s)

Oscar Flores <oflores@mmb.pcb.ub.es>

See Also

[peakDetection](#), [peakScoring](#), [plot](#),

Examples

```
#Generate a random peaks profile
reads = syntheticNucMap(nuc.len=40, lin.len=130)$syn.reads
cover = coverage(reads)

#Filter them
cover_fft = filterFFT(cover)

#Detect peaks
peaks = peakDetection(cover_fft, threshold="40%", score=TRUE, width=140)

#Plot peaks and coverage profile (show only a window)
plotPeaks(peaks, cover_fft, threshold="40%", start=1000, end=6000)
```

processReads

Process reads from High-Troughput Sequencing experiments

Description

This method allows the processment of NGS nucleosome reads from different sources and a basic manipulation of them. The tasks includes the correction of strand-specific single-end reads and the trimming of reads to a given length.

Usage

```
## S4 method for signature 'AlignedRead'
processReads(data, type = "single", fragmentLen, trim, ...)
## S4 method for signature 'RangedData'
processReads(data, type = "single", fragmentLen, trim, ...)
```

Arguments

<code>data</code>	Sequence reads objects, probably imported using other packages as <code>ShortRead</code> . Allowed object types are <code>AlignedRead</code> and <code>RangedData</code> with a <code>strand</code> attribute.
<code>type</code>	Describes the type of reads. Values allowed are <code>single</code> for single-ended reads and <code>paired</code> for paired-ended.
<code>fragmentLen</code>	Expected original length of the sequenced fragments. See details.
<code>trim</code>	Length to trim the reads (or extend them if <code>trim > read length</code>)
<code>...</code>	Other parameters passed to <code>fragmentLenDetect</code> if no fixed <code>fragmentLen</code> is given.

Details

This function reads a `AlignedRead` or a `RangedData` object containing the position, length and strand of the sequence reads.

It allows the processment of both paired and single ended reads. In the case of single end reads this function corrects the strand-specific mapping by shifting plus strand reads and minus strand reads towards a middle position where both strands are overlaped. This is done by accounting the expected fragment length (`fragmentLen`).

For paired end reads, mononucleosomal reads could extend more than expected length due to mapping issues or experimental conditions. In this case, the `fragmentLen` variable sets the threshold from which reads longer than it should be ignored.

If no value is supplied for `fragmentLen` it will be calculated automatically (increasing the computing time) using `fragmentLenDetect` with default parameters. Performance can be increased by tuning `fragmentLenDetect` parameters in a separated call and passing its result as `fragmentLen` parameter.

In some cases, could be useful trim the reads to a shorter length to improve the detection of nucleosome dyads, easing its detection and automatic positioning. The parameter `trim` allows the selection of how many nucleotides select from each read.

A special case for single-ended data is setting the `trim` to the same value as `fragmentLen`, so the reads will be extended strand-wise towards the 3' direction, creating an artificial map comparable with paired-ended data. The same but opposite can be performed with paired-end data, setting a `trim` value equal to the read length from paired ended, so paired-ended data will look like single-ended..

Value

`RangedData` containing the aligned/trimmed individual reads

Note

IMPORTANT: this information is only used to correct possible strand-specific mapping, this package doesn't link the two ends of paired reads.

Author(s)

Oscar Flores <oflores@mmb.pcb.ub.es>

See Also

[AlignedRead](#), [RangedData](#), [fragmentLenDetect](#)

Examples

```
#Load data
data(nucleosome_htseq)

#Process nucleosome reads, select only those shorter than 200bp
pr1 = processReads(nucleosome_htseq, fragmentLen=200)

#Now process them, but picking only the 40 bases surrounding the dyad
pr2 = processReads(nucleosome_htseq, fragmentLen=200, trim=40)

#Compare the results:
par(mfrow=c(2,1), mar=c(3,4,1,1))
plot(as.vector(coverage(pr1)[["chr1"]]), type="l", ylab="coverage (original)")
plot(as.vector(coverage(pr2)[["chr1"]]), type="l", ylab="coverage (trimmed)")
```

processTilingArray *Obtain and clean nucleosome positioning data from tiling array*

Description

Process and transform the microarray data coming from tiling array nucleosome positioning experiments.

Usage

```
processTilingArray(data, exprName, chrPattern, inferLen = 50, mc.cores = 1, quiet=FALSE)
```

Arguments

data	ExpressionSet object wich contains the data of the tiling array.
exprName	Name of the sample in ExpressionSet which contains the ratio between nucleosomal and genomic dna (if using Starr, the description argument supplied to getRatio function). If this name is not provided, it is assumed data has only one column.
chrPattern	Only chromosomes that contain chrPattern string will be selected from ExpressionSet. Sometimes tiling arrays contain control quality information that is imported as a chromosome. This allows filtering it. If no value is supplied, all chromosomes will be used.
inferLen	Maximum length (in basepairs) for allowing data gaps inference. See details for further information.
mc.cores	Number of cores available to parallel data processing.
quiet	Avoid printing on going information (TRUE FALSE)

Details

The processing of tiling arrays could be complicated as many types exists on the market. This function deals ok with Affymetrix Tiling Arrays in yeast, but hasn't been tested on other species or platforms.

The main aim is convert the output of preprocessing steps (supplied by third-parties packages) to a clean genome wide nucleosome occupancy profile.

Tiling arrays doesn't use to provide a one-basepair resolution data, so one gets one value per probe in the array, covering X basepairs and shifted (tiled) Y basepairs respect the surrounding ones. So, one gets a piece of information every Y basepairs.

This function tries to convert this noisy, low resolution data, to a one-basepair signal, which allows a fast recognition of nucleosomes without using large and artificious statistical machinery as Hidden Markov Models using posterior noise cleaning process.

As example, imagine your array has probes of 20mers and a tiling between probes of 10bp. Starting at position 1 (covering coordinates from 1 to 20), the next probe will be in position 10 (covering the coordinates 10 to 29). This can be represented as two hybridization intensity values on coordinates 1 and 10. This function will try to infer (using a lineal distribution) the values from 2 to 9 using the existing values of probes in coordinate 1 and coordinate 10.

The tiling space between adjacent array probes could be not constant, or could be also there are regions not covered in the used microarray. With the function argument inferLen you can specify wich amount of space (in basepairs) you allow to infer the non-present values.

If at some point the range not covered (gap) between two adjacent probes of the array is greater than inferLen value, then the coordinates between these probes will be setted to NA.

Value

RleList with the observed/inferred values for each coordinate.

Warning

This function could not cover all kind of arrays in the market. This package assumes the data is processed and normalized prior this processing, using standard microarray packages existing for R, like Starr.

Note

This function should be suitable for all data objects of kind ExpressionSet coding the annotations "chr" for chromosome and "pos" for position (accessible by pData(data@featureData)) and a expression value (accessible by exprs(data))

Author(s)

Oscar Flores <oflores@mmb.pcb.ub.es>

See Also

[ExpressionSet](#), [getRatio](#)

Examples

```
## Not run:
# Dataset cannot be provided for size restrictions
# This is the code used to get the hybridization ratio with Starr from CEL files
library("Starr")
TA_parsed <- readCelFile(BPMap, CELfiles, CELnames, CELtype,
                        featureData=TRUE, log.it=TRUE)
TA_loess <- normalize.Probes(TA_parsed, method="loess")
TA_ratio <- getRatio(TA_loess, TA_loess$type=="IP",
                    TA_loess$type=="CONTROL", "myRatio")

# From here, we use nucleR:

# Preprocess the array, using the calculated ratio feature we named
# "myRatio".

# This will also select only those chromosomes with the pattern
# "Sc:Oct_2003;chr", removing control data present in that tiling array.

# Finally, we allow that loci not covered by a probe being inferred
# from adjacent ones, as far as they are separated by 50bp or less
arr <- processTilingArray(TA_ratio, "myRatio",
                          chrPattern="Sc:Oct_2003;chr", inferLen=50)

# From here we can proceed with the analysis:
arr_fft <- filterFFT(arr)
arr_pea <- peakDetection(arr_fft)
plotPeaks(arr_pea, arr_fft)

## End(Not run)
```

readBAM	<i>Import reads from a list of BAM files.</i>
---------	---

Description

This function allows to load reads from BAM files from both single and paired-end commming from Next Generation Sequencing nucleosome mapping experiments.

Usage

```
readBAM(file, type = "single", mc.cores = 1)
```

Arguments

file	List of input BAM files.
type	Describes the type of reads. Values allowed are single for single-ended reads and paired for pair-ended.
mc.cores	If multicore support, the number of cores available.

Value

List of GRanges containing the reads of each input BAM file.

Author(s)

Oscar Flores <oflores@mmb.pcb.ub.es>, Ricard Illa <ricard.illa@irbbbarcelona.org>

syntheticNucMap	<i>Generates a synthetic nucleosome map</i>
-----------------	---

Description

This function generates a synthetic nucleosome map using the parameters given by the user and returns the coverage (like NGS experiments) or a pseudo-hybridization ratio (like Tiling Arrays) together with the perfect information about the well positioned and fuzzy nucleosome positions.

Usage

```
syntheticNucMap(wp.num=100, wp.del=10, wp.var=20, fuz.num=50, fuz.var=50, max.cover=20, nuc.len=147,
```

Arguments

<code>wp.num</code>	Number of well-positioned (non overlapped) nucleosomes. They are placed uniformly every <code>nuc.len+lin.len</code> basepairs.
<code>wp.del</code>	Number of well-positioned nucleosomes (the ones generated by <code>wp.num</code>) to remove. This will create an uncovered region.
<code>wp.var</code>	Maximum variance in basepairs of the well-positioned nucleosomes. This will create some variation in the position of the reads describing the same nucleosome
<code>fuz.num</code>	Number of fuzzy nucleosomes. They are distributed randomly over all the region. They could be overlapped with other well-positioned or fuzzy nucleosomes.
<code>fuz.var</code>	Maximum variance of the fuzzy nucleosomes. This allow to set different variance in well-positioned and fuzzy nucleosome reads (using <code>wp.var</code> and <code>fuz.var</code>)
<code>max.cover</code>	Maximum coverage of a nucleosome, i.e., how many times a nucleosome read can be repeated. The final coverage probably will be higher by the addition of overlapping nucleosomes.
<code>nuc.len</code>	Nucleosome length. It's not recommended change the default 147bp value.
<code>lin.len</code>	Linker DNA length. Usually around 20 bp.
<code>rnd.seed</code>	As this model uses random distributions for the placement, by setting the <code>rnd.seed</code> to a known value allows to reproduce maps in different executions or computers. If you don't need this, just left it in default value.
<code>as.ratio</code>	If <code>as.ratio=TRUE</code> this will create and return a synthetic naked DNA control map and the ratio between it and the nucleosome coverage. This can be used to simulate hybridization ratio data, like the one in Tiling Arrays.
<code>show.plot</code>	If TRUE, will plot the output coverage map, with the nucleosome calls and optionally the calculated ratio.
<code>...</code>	Additional parameters to be passed to plot if <code>show.plot=TRUE</code>

Value

A list with the following elements:

<code>wp.starts</code>	Start points of well-positioned nucleosomes
<code>wp.nreads</code>	Number of repetitions of each well positioned read
<code>wp.reads</code>	Well positioned nucleosome reads (IRanges format), containing the repetitions
<code>fuz.starts</code>	Start points of the fuzzy nucleosomes
<code>fuz.nreads</code>	Number of repetitions of each fuzzy nucleosome read
<code>fuz.reads</code>	Fuzzy nucleosome reads (IRanges format), containing all the repetitions
<code>syn.reads</code>	All synthetic nucleosome reads together (IRanges format)

The following elements will be only returned if `as.ratio=TRUE`:

<code>ctr.reads</code>	The pseudo-naked DNA (control) reads (IRanges format)
<code>syn.ratio</code>	The calculated ratio nucleosomal/control (R1e format)

Author(s)

Oscar Flores <oflores@mmb.pcb.ub.es>

Examples

```
#Generate a synthetic map with 50wp + 20fuzzy nucleosomes using fixed random seed=1
res = syntheticNucMap(wp.num=50, fuz.num=20, show.plot=TRUE, rnd.seed=1)

#Increase the fuzzyness
res = syntheticNucMap(wp.num=50, fuz.num=20, wp.var=70, fuz.var=150, show.plot=TRUE, rnd.seed=1)

#Calculate also a random map and get the ratio between random and nucleosomal
res = syntheticNucMap(wp.num=50, wp.del=0, fuz.num=20, as.ratio=TRUE, show.plot=TRUE, rnd.seed=1)

print(res)

#Different reads can be accessed separately from results
#Let's use this to plot the nucleosomal + the random map
par(mfrow=c(3,1), mar=c(3,4,1,1))
plot(as.vector(coverage(res$syn.reads)), type="h", col="red", ylab="nucleosomal", ylim=c(0,35))
plot(as.vector(coverage(res$ctr.reads)), type="h", col="blue", ylab="random", ylim=c(0,35))
plot(as.vector(res$syn.ratio), type="h", col="orange", ylab="ratio")
```

Index

- *Topic **attribute**
 - fragmentLenDetect, 10
 - pcKeepCompDetect, 15
- *Topic **datagen**
 - syntheticNucMap, 26
- *Topic **datasets**
 - nucleosome_htseq, 13
 - nucleosome_tiling, 14
- *Topic **file**
 - export.bed, 6
 - export.wig, 7
 - readBAM, 26
- *Topic **hplot**
 - plotPeaks, 20
- *Topic **manip**
 - controlCorrection, 3
 - coverage.rpm, 5
 - filterFFT, 8
 - mergeCalls, 12
 - peakDetection, 16
 - peakScoring, 18
 - processReads, 21
 - processTilingArray, 23
- *Topic **package**
 - nucleR-package, 2
- *Topic
 - readBAM, 26
 - .export.bed (export.bed), 6
- AlignedRead, 23
- controlCorrection, 3
- controlCorrection, list-method (controlCorrection), 3
- controlCorrection, numeric-method (controlCorrection), 3
- controlCorrection, Rle-method (controlCorrection), 3
- controlCorrection, SimpleRleList-method (controlCorrection), 3
- coverage, 5
- coverage.rpm, 5
- export.bed, 6
- export.bed, CompressedIRangesList-method (export.bed), 6
- export.bed, GRanges-method (export.bed), 6
- export.bed, IRanges-method (export.bed), 6
- export.bed, RangedData-method (export.bed), 6
- export.wig, 7
- ExpressionSet, 25
- filterFFT, 8, 17
- filterFFT, list-method (filterFFT), 8
- filterFFT, numeric-method (filterFFT), 8
- filterFFT, Rle-method (filterFFT), 8
- filterFFT, SimpleRleList-method (filterFFT), 8
- fragmentLenDetect, 10, 23
- fragmentLenDetect, AlignedRead-method (fragmentLenDetect), 10
- fragmentLenDetect, RangedData-method (fragmentLenDetect), 10
- getRatio, 25
- mergeCalls, 12
- nucleosome_htseq, 13
- nucleosome_tiling, 14
- nucleR (nucleR-package), 2
- nucleR-package, 2
- pcKeepCompDetect, 15
- peakDetection, 16, 19–21
- peakDetection, list-method (peakDetection), 16

peakDetection, numeric-method
 (peakDetection), 16

peakScoring, 13, 17, 18, 20, 21

peakScoring, IRanges-method
 (peakScoring), 18

peakScoring, IRangesList-method
 (peakScoring), 18

peakScoring, list-method (peakScoring),
 18

peakScoring, numeric-method
 (peakScoring), 18

plot, 21

plotPeaks, 20

plotPeaks, data.frame-method
 (plotPeaks), 20

plotPeaks, IRanges-method (plotPeaks), 20

plotPeaks, numeric-method (plotPeaks), 20

plotPeaks, RangedData-method
 (plotPeaks), 20

processReads, 5, 19, 21

processReads, AlignedRead-method
 (processReads), 21

processReads, RangedData-method
 (processReads), 21

processTilingArray, 23

RangedData, 23

readBAM, 26

syntheticNucMap, 26