Package 'AneuFinder'

October 11, 2016

Type Package

Title Analysis of Copy Number Variation in Single-Cell-Sequencing Data

Version 1.0.3

Date 2016-05

Author Aaron Taudt, Bjorn Bakker, David Porubsky

Maintainer Aaron Taudt <aaron.taudt@gmail.com>

Description This package implements functions for CNV calling, plotting, export and analysis from whole-genome single cell sequencing data.

Depends R (>= 3.3.0), GenomicRanges, cowplot, AneuFinderData

Imports utils, grDevices, graphics, stats, foreach, doParallel, BiocGenerics, S4Vectors, GenomeInfoDb, IRanges, Rsamtools, Biostrings, GenomicAlignments, preseqR, ggplot2, reshape2, ggdendro, ReorderCluster, mclust

Suggests knitr, testthat, BSgenome.Hsapiens.UCSC.hg19, BSgenome.Mmusculus.UCSC.mm10

License Artistic-2.0

LazyLoad yes

VignetteBuilder knitr

biocViews Software, CopyNumberVariation, GenomicVariation, HiddenMarkovModel, WholeGenome

URL https://github.com/ataudt/aneufinder.git

RoxygenNote 5.0.1

NeedsCompilation yes

R topics documented:

AneuFinder-package *Copy-number detection in WGSCS and Strand-Seq data*

Description

CNV detection in whole-genome single cell sequencing (WGSCS) and Strand-seq data using a Hidden Markov Model. The package implements CNV detection, commonly used plotting functions, export to BED format for upload to genome browsers, and measures for assessment of karyotype heterogeneity and quality metrics.

Details

The main function of this package is `Aneufinder` and produces several plots and browser files. If you want to have more fine-grained control over the different steps (binning, GC-correction, HMM, plotting) check the vignette Introduction to AneuFinder.

Author(s)

Aaron Taudt, David Porubsky

aneuBiHMM *Bivariate Hidden Markov Model*

Description

The `aneuBiHMM` object is output of the function `findSCEs` and is basically a list with various entries. The class() attribute of this list was set to "aneuBiHMM". For a given hmm, the entries can be accessed with the list operators 'hmm[[]]' and 'hmm$'.

Value

`ID`	An identifier that is used in various **AneuFinder** functions.
`bins`	A GRanges object containing the genomic bin coordinates, their read count and state classification.
`segments`	A GRanges object containing regions and their state classification.
`weights`	Weight for each component.
`transitionProbs`	Matrix of transition probabilities from each state (row) into each state (column).
`transitionProbs.initial`	Initial `transitionProbs` at the beginning of the Baum-Welch.
`startProbs`	Probabilities for the first bin
`startProbs.initial`	Initial `startProbs` at the beginning of the Baum-Welch.

`distributions` Estimated parameters of the emission distributions.

`distributions.initial`
Distribution parameters at the beginning of the Baum-Welch.

`convergenceInfo`
Contains information about the convergence of the Baum-Welch algorithm.

`convergenceInfo$eps`
Convergence threshold for the Baum-Welch.

`convergenceInfo$loglik`
Final loglikelihood after the last iteration.

`convergenceInfo$loglik.delta`
Change in loglikelihood after the last iteration (should be smaller than `eps`)

`convergenceInfo$num.iterations`
Number of iterations that the Baum-Welch needed to converge to the desired `eps`.

`convergenceInfo$time.sec`
Time in seconds that the Baum-Welch needed to converge to the desired `eps`.

See Also

findSCEs

`Aneufinder` *Wrapper function for the* `AneuFinder` *package*

Description

This function is an easy-to-use wrapper to bin the data, find copy-number-variations, find sister-chromatid-exchange events, plot genomewide heatmaps, distributions, profiles and karyograms.

Usage

```
Aneufinder(inputfolder, outputfolder, format, configfile = NULL, numCPU = 1,
  reuse.existing.files = TRUE, binsizes = 1e+06,
  variable.width.reference = NULL, reads.per.bin = NULL,
  pairedEndReads = FALSE, stepsize = NULL, assembly = NULL,
  chromosomes = NULL, remove.duplicate.reads = TRUE, min.mapq = 10,
  blacklist = NULL, correction.method = NULL, GC.BSgenome = NULL,
  mappability.reference = NULL, method = "univariate", eps = 0.1,
  max.time = 60, max.iter = 5000, num.trials = 15,
  states = c("zero-inflation", paste0(0:10, "-somy")),
  most.frequent.state.univariate = "2-somy",
  most.frequent.state.bivariate = "1-somy", resolution = c(3, 6),
  min.segwidth = 2, min.reads = 50, bw = 4 * binsizes[1], pval = 1e-08,
  cluster.plots = TRUE)
```

Arguments

inputfolder	Folder with either BAM or BED files.
outputfolder	Folder to output the results. If it does not exist it will be created.
format	Either 'bam' or 'bed', depending if your inputfolder contains files in BAM or BED format.
configfile	A file specifying the parameters of this function (without inputfolder, outputfolder and configfile). Having the parameters in a file can be handy if many samples with the same parameter settings are to be run. If a configfile is specified, it will take priority over the command line parameters.
numCPU	The numbers of CPUs that are used. Should not be more than available on your machine.
reuse.existing.files	A logical indicating whether or not existing files in outputfolder should be reused.
binsizes	An integer vector with bin sizes. If more than one value is given, output files will be produced for each bin size.
variable.width.reference	A BAM file that is used as reference to produce variable width bins. See variableWidthBins for details.
reads.per.bin	Approximate number of desired reads per bin. The bin size will be selected accordingly. Output files are produced for each value.
pairedEndReads	Set to TRUE if you have paired-end reads in your BAM files (not implemented for BED files).
stepsize	Fraction of the binsize that the sliding window is offset at each step. Example: If stepsize=0.1 and binsizes=c(200000,500000), the actual stepsize in basepairs is 20000 and 50000, respectively.
assembly	Please see fetchExtendedChromInfoFromUCSC for available assemblies. Only necessary when importing BED files. BAM files are handled automatically. Alternatively a data.frame generated by fetchExtendedChromInfoFromUCSC.
chromosomes	If only a subset of the chromosomes should be imported, specify them here.
remove.duplicate.reads	A logical indicating whether or not duplicate reads should be removed.
min.mapq	Minimum mapping quality when importing from BAM files. Set min.mapq=NULL to keep all reads.
blacklist	A GRanges or a bed(.gz) file with blacklisted regions. Reads falling into those regions will be discarded.
correction.method	Correction methods to be used for the binned read counts. Currently any combination of c('GC','mappability').
GC.BSgenome	A BSgenome object which contains the DNA sequence that is used for the GC correction.
mappability.reference	A file that serves as reference for mappability correction. Has to be the same format as specified by format.

method	Any combination of c('univariate','bivariate'). Option 'univariate' treats both strands as one, while option 'bivariate' treats both strands separately. NOTE: SCEs can only be called when method='bivariate'.
eps	Convergence threshold for the Baum-Welch algorithm.
max.time	The maximum running time in seconds for the Baum-Welch algorithm. If this time is reached, the Baum-Welch will terminate after the current iteration finishes. The default -1 is no limit.
max.iter	The maximum number of iterations for the Baum-Welch algorithm. The default -1 is no limit.
num.trials	The number of trials to find a fit where state most.frequent.state is most frequent. Each time, the HMM is seeded with different random initial values.
states	A subset or all of c("zero-inflation","0-somy","1-somy","2-somy","3-somy","4-somy",...). This vector defines the states that are used in the Hidden Markov Model. The order of the entries must not be changed.
most.frequent.state.univariate	One of the states that were given in states. The specified state is assumed to be the most frequent one when running the univariate HMM. This can help the fitting procedure to converge into the correct fit. Default is '2-somy'.
most.frequent.state.bivariate	One of the states that were given in states. The specified state is assumed to be the most frequent one when running the bivariate HMM. This can help the fitting procedure to converge into the correct fit. Default is '1-somy'.
resolution	An integer vector specifying the resolution at bin level at which to scan for SCE events.
min.segwidth	Minimum segment length in bins when scanning for SCE events.
min.reads	Minimum number of reads required for SCE refinement.
bw	Bandwidth for SCE hotspot detection (see hotspotter for further details).
pval	P-value for SCE hotspot detection (see hotspotter for further details).
cluster.plots	A logical indicating whether plots should be clustered by similarity.

Value

NULL

Author(s)

Aaron Taudt

Examples

```
## Not run:
## The following call produces plots and genome browser files for all BAM files in "my-data-folder"
Aneufinder(inputfolder="my-data-folder", outputfolder="my-output-folder", format='bam')
## End(Not run)
```

aneuHMM *Hidden Markov Model*

Description

The aneuHMM object is output of the function findCNVs and is basically a list with various entries. The class() attribute of this list was set to "aneuHMM". For a given hmm, the entries can be accessed with the list operators 'hmm[[]]' and 'hmm$'.

Value

ID
: An identifier that is used in various **AneuFinder** functions.

bins
: A GRanges object containing the genomic bin coordinates, their read count and state classification.

segments
: A GRanges object containing regions and their state classification.

weights
: Weight for each component.

transitionProbs
: Matrix of transition probabilities from each state (row) into each state (column).

transitionProbs.initial
: Initial transitionProbs at the beginning of the Baum-Welch.

startProbs
: Probabilities for the first bin

startProbs.initial
: Initial startProbs at the beginning of the Baum-Welch.

distributions
: Estimated parameters of the emission distributions.

distributions.initial
: Distribution parameters at the beginning of the Baum-Welch.

convergenceInfo
: Contains information about the convergence of the Baum-Welch algorithm.

convergenceInfo$eps
: Convergence threshold for the Baum-Welch.

convergenceInfo$loglik
: Final loglikelihood after the last iteration.

convergenceInfo$loglik.delta
: Change in loglikelihood after the last iteration (should be smaller than eps)

convergenceInfo$num.iterations
: Number of iterations that the Baum-Welch needed to converge to the desired eps.

convergenceInfo$time.sec
: Time in seconds that the Baum-Welch needed to converge to the desired eps.

See Also

findCNVs

bam2GRanges *Import BAM file into GRanges*

Description

Import aligned reads from a BAM file into a GRanges object.

Usage

```
bam2GRanges(bamfile, bamindex = bamfile, chromosomes = NULL,
  pairedEndReads = FALSE, remove.duplicate.reads = FALSE, min.mapq = 10,
  max.fragment.width = 1000, blacklist = NULL, what = "mapq")
```

Arguments

bamfile	A sorted BAM file.
bamindex	BAM index file. Can be specified without the .bai ending. If the index file does not exist it will be created and a warning is issued.
chromosomes	If only a subset of the chromosomes should be imported, specify them here.
pairedEndReads	Set to TRUE if you have paired-end reads in your BAM files (not implemented for BED files).
remove.duplicate.reads	A logical indicating whether or not duplicate reads should be removed.
min.mapq	Minimum mapping quality when importing from BAM files. Set min.mapq=NULL to keep all reads.
max.fragment.width	Maximum allowed fragment length. This is to filter out erroneously wrong fragments due to mapping errors of paired end reads.
blacklist	A GRanges or a bed(.gz) file with blacklisted regions. Reads falling into those regions will be discarded.
what	A character vector of fields that are returned. Type scanBamWhat to see what is available.

Value

A GRanges object containing the reads.

Examples

```
## Get an example BAM file with single-cell-sequencing reads
bamfile <- system.file("extdata", "BB150803_IV_074.bam", package="AneuFinderData")
## Read the file into a GRanges object
reads <- bam2GRanges(bamfile, chromosomes=c(1:19,'X','Y'), pairedEndReads=FALSE,
                     min.mapq=10, remove.duplicate.reads=TRUE)
print(reads)
```

bed2GRanges *Import BED file into GRanges*

Description

Import aligned reads from a BED file into a GRanges object.

Usage

```
bed2GRanges(bedfile, assembly, chromosomes = NULL,
  remove.duplicate.reads = FALSE, min.mapq = 10,
  max.fragment.width = 1000, blacklist = NULL)
```

Arguments

bedfile	A file with aligned reads in BED format. The columns have to be c('chromosome','start','end','description
assembly	Please see fetchExtendedChromInfoFromUCSC for available assemblies. Only necessary when importing BED files. BAM files are handled automatically. Alternatively a data.frame generated by fetchExtendedChromInfoFromUCSC.
chromosomes	If only a subset of the chromosomes should be imported, specify them here.
remove.duplicate.reads	A logical indicating whether or not duplicate reads should be removed.
min.mapq	Minimum mapping quality when importing from BAM files. Set min.mapq=NULL to keep all reads.
max.fragment.width	Maximum allowed fragment length. This is to filter out erroneously wrong fragments.
blacklist	A GRanges or a bed(.gz) file with blacklisted regions. Reads falling into those regions will be discarded.

Value

A GRanges object containing the reads.

Examples

```
## Get an example BED file with single-cell-sequencing reads
bedfile <- system.file("extdata", "KK150311_VI_07.bam.bed.gz", package="AneuFinderData")
## Read the file into a GRanges object
reads <- bed2GRanges(bedfile, assembly='mm10', chromosomes=c(1:19,'X','Y'),
                     min.mapq=10, remove.duplicate.reads=TRUE)
print(reads)
```

binned.data *Binned read counts*

Description

A GRanges object which contains binned read counts as meta data column reads. It is output of the various binning functions.

binning *Bin the genome*

Description

Please see functions fixedWidthBins and variableWidthBins for further details.

binReads *Convert aligned reads from various file formats into read counts in equidistant bins*

Description

Convert aligned reads in .bam or .bed(.gz) format into read counts in equidistant windows.

Usage

```
binReads(file, format, assembly, ID = basename(file), bamindex = file,
  chromosomes = NULL, pairedEndReads = FALSE, min.mapq = 10,
  remove.duplicate.reads = TRUE, max.fragment.width = 1000,
  blacklist = NULL, outputfolder.binned = "binned_data", binsizes = 1e+06,
  reads.per.bin = NULL, bins = NULL, variable.width.reference = NULL,
  stepsize = NULL, save.as.RData = FALSE, calc.complexity = TRUE,
  call = match.call(), reads.store = FALSE, outputfolder.reads = "data",
  reads.return = FALSE, reads.overwrite = FALSE, reads.only = FALSE)
```

Arguments

file	A file with aligned reads. Alternatively a GRanges with aligned reads if format is set to 'GRanges'.
format	One of c('bam', 'bed', 'GRanges').
assembly	Please see fetchExtendedChromInfoFromUCSC for available assemblies. Only necessary when importing BED files. BAM files are handled automatically. Alternatively a data.frame generated by fetchExtendedChromInfoFromUCSC.

`ID` An identifier that will be used to identify the file throughout the workflow and in plotting.

`bamindex` BAM index file. Can be specified without the .bai ending. If the index file does not exist it will be created and a warning is issued.

`chromosomes` If only a subset of the chromosomes should be binned, specify them here.

`pairedEndReads` Set to `TRUE` if you have paired-end reads in your BAM files (not implemented for BED files).

`min.mapq` Minimum mapping quality when importing from BAM files. Set `min.mapq=NULL` to keep all reads.

`remove.duplicate.reads`
A logical indicating whether or not duplicate reads should be removed.

`max.fragment.width`
Maximum allowed fragment length. This is to filter out erroneously wrong fragments due to mapping errors of paired end reads.

`blacklist` A `GRanges` or a bed(.gz) file with blacklisted regions. Reads falling into those regions will be discarded.

`outputfolder.binned`
Folder to which the binned data will be saved. If the specified folder does not exist, it will be created.

`binsizes` An integer vector with bin sizes. If more than one value is given, output files will be produced for each bin size.

`reads.per.bin` Approximate number of desired reads per bin. The bin size will be selected accordingly. Output files are produced for each value.

`bins` A named `list` with `GRanges` containing precalculated bins produced by `fixedWidthBins` or `variableWidthBins`. Names must correspond to the binsize.

`variable.width.reference`
A BAM file that is used as reference to produce variable width bins. See `variableWidthBins` for details.

`stepsize` Fraction of the binsize that the sliding window is offset at each step. Example: If `stepsize=0.1` and `binsizes=c(200000,500000)`, the actual stepsize in basepairs is 20000 and 50000, respectively. NOT USED AT THE MOMENT.

`save.as.RData` If set to `FALSE`, no output file will be written. Instead, a GenomicRanges object containing the binned data will be returned. Only the first binsize will be processed in this case.

`calc.complexity`
A logical indicating whether or not to estimate library complexity.

`call` The `match.call()` of the parent function.

`reads.store` If `TRUE` processed read fragments will be saved to file. Reads are processed according to `min.mapq` and `remove.duplicate.reads`. Paired end reads are coerced to single end fragments.

`outputfolder.reads`
Folder to which the read fragments will be saved. If the specified folder does not exist, it will be created.

`reads.return` If `TRUE` no binning is done and instead, read fragments from the input file are returned in `GRanges` format.

`reads.overwrite` Whether or not an existing file with read fragments should be overwritten.

`reads.only` If `TRUE` only read fragments are stored and/or returned and no binning is done.

Details

Convert aligned reads from .bam or .bed(.gz) files into read counts in equidistant windows (bins). This function uses `countOverlaps` to calculate the read counts.

Value

The function produces a `list()` of GRanges objects with one meta data column 'reads' that contains the read count. This binned data will be either written to file (`save.as.RData=FALSE`) or given as return value (`save.as.RData=FALSE`).

See Also

binning

Examples

```
## Get an example BED file with single-cell-sequencing reads
bedfile <- system.file("extdata", "KK150311_VI_07.bam.bed.gz", package="AneuFinderData")
## Bin the BED file into bin size 1Mb
binned <- binReads(bedfile, format='bed', assembly='mm10', binsize=1e6,
                   chromosomes=c(1:19,'X','Y'))
print(binned)
```

---

bivariate.findCNVs *Find copy number variations (bivariate)*

Description

`bivariate.findCNVs` finds CNVs using read count information from both strands.

Usage

```
bivariate.findCNVs(binned.data, ID = NULL, eps = 0.1, init = "standard",
  max.time = -1, max.iter = -1, num.trials = 1, eps.try = NULL,
  num.threads = 1, count.cutoff.quantile = 0.999,
  states = c("zero-inflation", paste0(0:10, "-somy")),
  most.frequent.state = "1-somy", algorithm = "EM", initial.params = NULL)
```

Arguments

binned.data	A GRanges object with binned read counts.
ID	An identifier that will be used to identify this sample in various downstream functions. Could be the file name of the binned.data for example.
eps	Convergence threshold for the Baum-Welch algorithm.
init	One of the following initialization procedures: standard The negative binomial of state '2-somy' will be initialized with mean=mean(counts), var=var(counts). This procedure usually gives good convergence. random Mean and variance of the negative binomial of state '2-somy' will be initialized with random values (in certain boundaries, see source code). Try this if the standard procedure fails to produce a good fit.
max.time	The maximum running time in seconds for the Baum-Welch algorithm. If this time is reached, the Baum-Welch will terminate after the current iteration finishes. The default -1 is no limit.
max.iter	The maximum number of iterations for the Baum-Welch algorithm. The default -1 is no limit.
num.trials	The number of trials to find a fit where state most.frequent.state is most frequent. Each time, the HMM is seeded with different random initial values.
eps.try	If code num.trials is set to greater than 1, eps.try is used for the trial runs. If unset, eps is used.
num.threads	Number of threads to use. Setting this to >1 may give increased performance.
count.cutoff.quantile	A quantile between 0 and 1. Should be near 1. Read counts above this quantile will be set to the read count specified by this quantile. Filtering very high read counts increases the performance of the Baum-Welch fitting procedure. However, if your data contains very few peaks they might be filtered out. Set count.cutoff.quantile=1 in this case.
states	A subset or all of c("zero-inflation","0-somy","1-somy","2-somy","3-somy","4-somy",...). This vector defines the states that are used in the Hidden Markov Model. The order of the entries must not be changed.
most.frequent.state	One of the states that were given in states or 'none'. The specified state is assumed to be the most frequent one. This can help the fitting procedure to converge into the correct fit.
algorithm	One of c('baumWelch','EM'). The expectation maximization ('EM') will find the most likely states and fit the best parameters to the data, the 'baumWelch' will find the most likely states using the initial parameters.
initial.params	A aneuHMM object or file containing such an object from which initial starting parameters will be extracted.

Value

An aneuBiHMM object.

`blacklist` *Make a blacklist for genomic regions*

Description

Produce a blacklist of genomic regions with a high ratio of duplicate to unique reads. This blacklist can be used to exclude reads for analysis in `Aneufinder`, `bam2GRanges` and `bed2GRanges`. This function produces a pre-blacklist which has to manually filtered with a sensible cutoff. See the examples section for details.

Usage

```
blacklist(files, format, assembly, bins, min.mapq = 10,
  pairedEndReads = FALSE)
```

Arguments

`files`	A list of either BAM or BED files.
`format`	The format of `files`. Either 'bam' or 'bed'.
`assembly`	Please see `fetchExtendedChromInfoFromUCSC` for available assemblies. Only necessary when importing BED files. BAM files are handled automatically. Alternatively a data.frame generated by `fetchExtendedChromInfoFromUCSC`.
`bins`	A list with one `GRanges` with binned read counts generated by `fixedWidthBins`.
`min.mapq`	Minimum mapping quality when importing from BAM files. Set `min.mapq=NULL` to keep all reads.
`pairedEndReads`	Set to `TRUE` if you have paired-end reads in your BAM files (not implemented for BED files).

Value

A `GRanges` with the same coordinates as `bins` with metadata columns ratio, duplicated counts and deduplicated counts.

Examples

```
## Get an example BAM file with single-cell-sequencing reads
bamfile <- system.file("extdata", "BB150803_IV_074.bam", package="AneuFinderData")
## Prepare the blacklist
bins <- fixedWidthBins(assembly='mm10', binsizes=1e6, chromosome.format='NCBI')
pre.blacklist <- blacklist(bamfile, format='bam', bins=bins)
## Plot a histogram to decide on a sensible cutoff
qplot(pre.blacklist$ratio, binwidth=0.1)
## Make the blacklist with cutoff = 1.9
blacklist <- pre.blacklist[pre.blacklist$ratio > 1.9]
```

clusterByQuality *Cluster based on quality variables*

Description

This function uses the **mclust** package to cluster the input samples based on various quality measures.

Usage

```
clusterByQuality(hmms, G = 1:9, itmax = c(100, 100),
  measures = c("spikiness", "entropy", "num.segments", "bhattacharyya",
  "loglik"), orderBy = "spikiness", reverseOrder = FALSE)
```

Arguments

hmms	A list of aneuHMM objects or a list of files that contain such objects.
G	An integer vector specifying the number of clusters that are compared. See Mclust for details.
itmax	The maximum number of outer and inner iterations for the Mclust function. See emControl for details.
measures	The quality measures that are used for the clustering. Supported is any combination of c('spikiness','entropy','num.segments','bhattacharyya','loglik','complexity','s
orderBy	The quality measure to order the clusters by. Default is 'spikiness'.
reverseOrder	Logical indicating whether the ordering by orderBy is reversed.

Details

The employed quality measures are:

- Spikiness
- Entropy
- Number of segments
- Bhattacharrya distance
- Loglikelihood

Value

A list with the classification, parameters and the Mclust fit.

Author(s)

Aaron Taudt

Examples

```
## Get a list of HMMs
folder <- system.file("extdata", "primary-lung", "hmms", package="AneuFinderData")
files <- list.files(folder, full.names=TRUE)
cl <- clusterByQuality(files)
## Plot the clustering and print the parameters
plot(cl$Mclust, what='classification')
print(cl$parameters)
## Select files from the best 2 clusters for further processing
best.files <- unlist(cl$classification[1:2])
```

`collapseBins` *Collapse consecutive bins*

Description

The function will collapse consecutive bins which have, for example, the same combinatorial state.

Usage

```
collapseBins(data, column2collapseBy = NULL, columns2sumUp = NULL,
  columns2average = NULL, columns2getMax = NULL, columns2drop = NULL)
```

Arguments

`data`	A data.frame containing the genomic coordinates in the first three columns.
`column2collapseBy`	The number of the column which will be used to collapse all other inputs. If a set of consecutive bins has the same value in this column, they will be aggregated into one bin with adjusted genomic coordinates. If `NULL` directly adjacent bins will be collapsed.
`columns2sumUp`	Column numbers that will be summed during the aggregation process.
`columns2average`	Column numbers that will be averaged during the aggregation process.
`columns2getMax`	Column numbers where the maximum will be chosen during the aggregation process.
`columns2drop`	Column numbers that will be dropped after the aggregation process.

Details

The following tables illustrate the principle of the collapsing:

Input data:

seqnames	start	end	column2collapseBy	moreColumns	columns2sumUp
chr1	0	199	2	1 10	1 3

chr1	200	399	2	2 11	0 3
chr1	400	599	2	3 12	1 3
chr1	600	799	1	4 13	0 3
chr1	800	999	1	5 14	1 3

Output data:

seqnames	start	end	column2collapseBy	moreColumns	columns2sumUp
chr1	0	599	2	1 10	2 9
chr1	600	999	1	4 13	1 6

Value

A data.frame.

Author(s)

Aaron Taudt

Examples

```
## Get an example BED file with single-cell-sequencing reads
bedfile <- system.file("extdata", "KK150311_VI_07.bam.bed.gz", package="AneuFinderData")
## Bin the BAM file into bin size 1Mp
binned <- binReads(bedfile, format='bed', assembly='mm10', binsize=1e6,
                  chromosomes=c(1:19,'X','Y'))
## Collapse the bins by chromosome and get average, summed and maximum read count
df <- as.data.frame(binned[[1]])
# Remove one bin for illustration purposes
df <- df[-3,]
head(df)
collapseBins(df, column2collapseBy='seqnames', columns2sumUp=c('width','counts'),
                       columns2average='counts', columns2getMax='counts',
                       columns2drop=c('mcounts','pcounts'))
collapseBins(df, column2collapseBy=NULL, columns2sumUp=c('width','counts'),
                       columns2average='counts', columns2getMax='counts',
                       columns2drop=c('mcounts','pcounts'))
```

`colors` **AneuFinder** *color scheme*

Description

Get the color schemes that are used in the AneuFinder plots.

Usage

```
stateColors(states = c("zero-inflation", paste0(0:10, "-somy"), "total"))

strandColors(strands = c("+", "-"))
```

Arguments

states	A character vector with states whose color should be returned.
strands	A character vector with strands whose color should be returned. Any combination of `c('+','-','*')`.

Value

A character vector with colors.

Functions

- `stateColors`: Colors that are used for the states.
- `strandColors`: Colors that are used to distinguish strands.

Examples

```
## Make a nice pie chart with the AneuFinder state color scheme
statecolors <- stateColors()
pie(rep(1,length(statecolors)), labels=names(statecolors), col=statecolors)

## Make a nice pie chart with the AneuFinder strand color scheme
strandcolors <- strandColors()
pie(rep(1,length(strandcolors)), labels=names(strandcolors), col=strandcolors)
```

---

correctGC *GC correction*

Description

Correct a list of `binned.data` by GC content.

Usage

```
correctGC(binned.data.list, GC.BSgenome, same.binsize = FALSE)
```

Arguments

binned.data.list	A list with binned.data objects or a list of filenames containing such objects.
GC.BSgenome	A BSgenome object which contains the DNA sequence that is used for the GC correction.
same.binsize	If TRUE the GC content will only be calculated once. Set this to TRUE if all binned.data objects describe the same genome at the same binsize.

Value

A list with binned.data objects with adjusted read counts.

Author(s)

Aaron Taudt

Examples

```
## Get a BED file, bin it and run GC correction
bedfile <- system.file("extdata", "KK150311_VI_07.bam.bed.gz", package="AneuFinderData")
binned <- binReads(bedfile, format='bed', assembly='mm10', binsize=1e6,
                   chromosomes=c(1:19,'X','Y'))
plot(binned[[1]], type=1)
if (require(BSgenome.Mmusculus.UCSC.mm10)) {
 binned.GC <- correctGC(list(binned[[1]]), GC.BSgenome=BSgenome.Mmusculus.UCSC.mm10)
 plot(binned.GC[[1]], type=1)
}
```

---

correctMappability *Mappability correction*

Description

Correct a list of binned.data by mappability.

Usage

```
correctMappability(binned.data.list, same.binsize, reference, format, assembly,
  pairedEndReads = FALSE, min.mapq = 10, remove.duplicate.reads = TRUE,
  max.fragment.width = 1000)
```

Arguments

binned.data.list	A list with binned.data objects or a list of filenames containing such objects.
same.binsize	If TRUE the mappability correction will only be calculated once. Set this to TRUE if all binned.data objects describe the same genome at the same binsize.
reference	A file or GRanges with aligned reads.
format	Format of the reference, one of c('bam','bed','GRanges').
assembly	Please see fetchExtendedChromInfoFromUCSC for available assemblies. Only necessary when importing BED files. BAM files are handled automatically. Alternatively a data.frame generated by fetchExtendedChromInfoFromUCSC.
pairedEndReads	Set to TRUE if you have paired-end reads in your BAM files (not implemented for BED files).
min.mapq	Minimum mapping quality when importing from BAM files. Set min.mapq=NULL to keep all reads.
remove.duplicate.reads	A logical indicating whether or not duplicate reads should be removed.
max.fragment.width	Maximum allowed fragment length. This is to filter out erroneously wrong fragments due to mapping errors of paired end reads.

Value

A list with binned.data objects with adjusted read counts.

Author(s)

Aaron Taudt

---

deltaWCalculator *Calculate deltaWs*

Description

This function will calculate deltaWs from a GRanges object with read fragments.

Usage

```
deltaWCalculator(frags, reads.per.window = 10)
```

Arguments

frags	A GRanges with read fragments (see bam2GRanges).
reads.per.window	Number of reads in each dynamic window.

Value

The input `frags` with additional meta-data columns.

Author(s)

Aaron Taudt, David Porubsky, Ashley Sanders

`estimateComplexity` *Estimate library complexity*

Description

Estimate library complexity using a very simple "Michaelis-Menten" approach and the sophisticated approach from the **preseqR** package.

Usage

```
estimateComplexity(reads)
```

Arguments

`reads`	A `GRanges` object with read fragments. NOTE: Complexity estimation relies on duplicate reads and therefore the duplicates have to be present in the input.

Value

A `list` with estimated complexity values and plots.

`export` *Export genome browser viewable files*

Description

Export copy-number-variation state or read counts as genome browser viewable file

Usage

```
exportCNVs(hmms, filename, cluster = TRUE, export.CNV = TRUE,
  export.SCE = TRUE)

exportReadCounts(hmms, filename)

exportGRanges(gr, filename, header = TRUE, trackname = NULL, score = NULL,
  priority = NULL, append = FALSE, chromosome.format = "UCSC")
```

Arguments

hmms	A list of aneuHMM objects or files that contain such objects.
filename	The name of the file that will be written. The appropriate ending will be appended, either ".bed.gz" for CNV-state or ".wig.gz" for read counts. Any existing file will be overwritten.
cluster	If TRUE, the samples will be clustered by similarity in their CNV-state.
export.CNV	A logical, indicating whether the CNV-state shall be exported.
export.SCE	A logical, indicating whether the SCE events shall be exported.
gr	A GRanges object.
header	A logical indicating whether the output file will have a heading track line (TRUE) or not (FALSE).
trackname	The name that will be used as track name and description in the header.
score	A vector of the same length as gr, which will be used for the 'score' column in the BED file.
priority	Priority of the track for display in the genome browser.
append	Append to filename.
chromosome.format	A character specifying the format of the chromosomes if assembly is specified. Either 'NCBI' for (1,2,3 ...) or 'UCSC' for (chr1,chr2,chr3 ...).#' @importFrom utils write.table

Details

Use exportCNVs to export the copy-number-variation state from an aneuHMM object in BED format. Use exportReadCounts to export the binned read counts from an aneuHMM object in WIGGLE format. Use exportGRanges to export a GRanges object in BED format.

Value

NULL

Functions

- exportCNVs: Export CNV-state as .bed.gz file
- exportReadCounts: Export binned read counts as .wig.gz file
- exportGRanges: Export GRanges object as BED file.

Author(s)

Aaron Taudt

Examples

```
## Not run:
## Get results from a small-cell-lung-cancer
folder <- system.file("extdata", "primary-lung", "hmms", package="AneuFinderData")
files <- list.files(folder, full.names=TRUE)
## Export the CNV states for upload to the UCSC genome browser
exportCNVs(files, filename='upload-me-to-a-genome-browser', cluster=TRUE)
## End(Not run)
```

`filterSegments` *Filter segments by minimal size*

Description

`filterSegments` filters out segments below a specified minimal segment size. This can be useful to get rid of boundary effects from the Hidden Markov approach.

Usage

```
filterSegments(segments, min.seg.width)
```

Arguments

`segments`	A GRanges object.
`min.seg.width`	The minimum segment width in base-pairs.

Value

The input `model` with adjusted segments.

Author(s)

Aaron Taudt

Examples

```
## Load an HMM
file <- list.files(system.file("extdata", "primary-lung", "hmms",
                  package="AneuFinderData"), full.names=TRUE)
hmm <- loadHmmsFromFiles(file)[[1]]
## Check number of segments before and after filtering
length(hmm$segments)
hmm$segments <- filterSegments(hmm$segments, min.seg.width=2*width(hmm$bins)[1])
length(hmm$segments)
```

findCNVs *Find copy number variations*

Description

findCNVs classifies the binned read counts into several states which represent copy-number-variation.

Usage

```
findCNVs(binned.data, ID = NULL, eps = 0.1, init = "standard",
  max.time = -1, max.iter = 1000, num.trials = 15, eps.try = 10 * eps,
  num.threads = 1, count.cutoff.quantile = 0.999, strand = "*",
  states = c("zero-inflation", paste0(0:10, "-somy")),
  most.frequent.state = "2-somy", method = "univariate", algorithm = "EM",
  initial.params = NULL)
```

Arguments

binned.data	A GRanges object with binned read counts.
ID	An identifier that will be used to identify this sample in various downstream functions. Could be the file name of the binned.data for example.
eps	Convergence threshold for the Baum-Welch algorithm.
init	One of the following initialization procedures: standard The negative binomial of state '2-somy' will be initialized with mean=mean(counts), var=var(counts). This procedure usually gives good convergence. random Mean and variance of the negative binomial of state '2-somy' will be initialized with random values (in certain boundaries, see source code). Try this if the standard procedure fails to produce a good fit.
max.time	The maximum running time in seconds for the Baum-Welch algorithm. If this time is reached, the Baum-Welch will terminate after the current iteration finishes. The default -1 is no limit.
max.iter	The maximum number of iterations for the Baum-Welch algorithm. The default -1 is no limit.
num.trials	The number of trials to find a fit where state most.frequent.state is most frequent. Each time, the HMM is seeded with different random initial values.
eps.try	If code num.trials is set to greater than 1, eps.try is used for the trial runs. If unset, eps is used.
num.threads	Number of threads to use. Setting this to >1 may give increased performance.
count.cutoff.quantile	A quantile between 0 and 1. Should be near 1. Read counts above this quantile will be set to the read count specified by this quantile. Filtering very high read counts increases the performance of the Baum-Welch fitting procedure. However, if your data contains very few peaks they might be filtered out. Set count.cutoff.quantile=1 in this case.

strand — Run the HMM only for the specified strand. One of `c('+', '-', '*')`.

states — A subset or all of `c("zero-inflation","0-somy","1-somy","2-somy","3-somy","4-somy",...)`. This vector defines the states that are used in the Hidden Markov Model. The order of the entries must not be changed.

most.frequent.state — One of the states that were given in `states` or 'none'. The specified state is assumed to be the most frequent one. This can help the fitting procedure to converge into the correct fit.

method — One of `c('univariate','bivariate')`. In the univariate case strand information is discarded, while in the bivariate case strand information is used for the fitting.

algorithm — One of `c('baumWelch','EM')`. The expectation maximization (`'EM'`) will find the most likely states and fit the best parameters to the data, the `'baumWelch'` will find the most likely states using the initial parameters.

initial.params — A aneuHMM object or file containing such an object from which initial starting parameters will be extracted.

Details

`findCNVs` uses a 6-state Hidden Markov Model to classify the binned read counts: state '0-somy' with a delta function as emission densitiy (only zero read counts), '1-somy','2-somy','3-somy','4-somy', etc. with negative binomials (see dnbinom) as emission densities. A Baum-Welch algorithm is employed to estimate the parameters of the distributions. See our paper `citation("AneuFinder")` for a detailed description of the method.

Value

An aneuHMM object.

Author(s)

Aaron Taudt

Examples

```
## Get an example BED file with single-cell-sequencing reads
bedfile <- system.file("extdata", "KK150311_VI_07.bam.bed.gz", package="AneuFinderData")
## Bin the BAM file into bin size 1Mp
binned <- binReads(bedfile, format='bed', assembly='mm10', binsize=1e6,
                   chromosomes=c(1:19,'X','Y'))
## Fit the Hidden Markov Model
model <- findCNVs(binned[[1]], eps=0.1, max.time=60)
## Check the fit
plot(model, type='histogram')
```

`findSCEs` *Find sister chromatid exchanges*

Description

`findSCEs` classifies the binned read counts into several states which represent the number of chromatids on each strand.

Usage

```
findSCEs(binned.data, ID = NULL, eps = 0.1, init = "standard",
  max.time = -1, max.iter = 1000, num.trials = 5, eps.try = 10 * eps,
  num.threads = 1, count.cutoff.quantile = 0.999, strand = "*",
  states = c("zero-inflation", paste0(0:10, "-somy")),
  most.frequent.state = "1-somy", algorithm = "EM", initial.params = NULL)
```

Arguments

`binned.data`	A GRanges object with binned read counts.
`ID`	An identifier that will be used to identify this sample in various downstream functions. Could be the file name of the `binned.data` for example.
`eps`	Convergence threshold for the Baum-Welch algorithm.
`init`	One of the following initialization procedures: `standard` The negative binomial of state '2-somy' will be initialized with `mean=mean(counts)`, `var=var(counts)`. This procedure usually gives good convergence. `random` Mean and variance of the negative binomial of state '2-somy' will be initialized with random values (in certain boundaries, see source code). Try this if the `standard` procedure fails to produce a good fit.
`max.time`	The maximum running time in seconds for the Baum-Welch algorithm. If this time is reached, the Baum-Welch will terminate after the current iteration finishes. The default -1 is no limit.
`max.iter`	The maximum number of iterations for the Baum-Welch algorithm. The default -1 is no limit.
`num.trials`	The number of trials to find a fit where state `most.frequent.state` is most frequent. Each time, the HMM is seeded with different random initial values.
`eps.try`	If code num.trials is set to greater than 1, `eps.try` is used for the trial runs. If unset, `eps` is used.
`num.threads`	Number of threads to use. Setting this to >1 may give increased performance.
`count.cutoff.quantile`	A quantile between 0 and 1. Should be near 1. Read counts above this quantile will be set to the read count specified by this quantile. Filtering very high read counts increases the performance of the Baum-Welch fitting procedure. However, if your data contains very few peaks they might be filtered out. Set `count.cutoff.quantile=1` in this case.

strand — Run the HMM only for the specified strand. One of `c('+', '-', '*')`.

states — A subset or all of `c("zero-inflation","0-somy","1-somy","2-somy","3-somy","4-somy",...)`. This vector defines the states that are used in the Hidden Markov Model. The order of the entries must not be changed.

most.frequent.state — One of the states that were given in `states` or 'none'. The specified state is assumed to be the most frequent one. This can help the fitting procedure to converge into the correct fit.

algorithm — One of `c('baumWelch','EM')`. The expectation maximization (`'EM'`) will find the most likely states and fit the best parameters to the data, the `'baumWelch'` will find the most likely states using the initial parameters.

initial.params — A aneuHMM object or file containing such an object from which initial starting parameters will be extracted.

Details

`findSCEs` uses a Hidden Markov Model to classify the binned read counts: state 'zero-inflation' with a delta function as emission densitiy (only zero read counts), '0-somy' with geometric distribution, '1-somy','2-somy','3-somy','4-somy', etc. with negative binomials (see dnbinom) as emission densities. A expectation-maximization (EM) algorithm is employed to estimate the parameters of the distributions. See our paper `citation("AneuFinder")` for a detailed description of the method.

Value

An aneuBiHMM object.

Author(s)

Aaron Taudt

Examples

```
## Get an example BED file with single-cell-sequencing reads
bedfile <- system.file("extdata", "KK150311_VI_07.bam.bed.gz", package="AneuFinderData")
## Bin the BAM file into bin size 1Mp
binned <- binReads(bedfile, format='bed', assembly='hg19', binsize=1e6,
                   chromosomes=c(1:22,'X','Y'), pairedEndReads=TRUE)
## Fit the Hidden Markov Model
model <- findSCEs(binned[[1]], eps=0.1, max.time=60)
## Check the fit
plot(model, type='histogram')
plot(model, type='profile')
```

`fixedWidthBins` *Make fixed-width bins*

Description

Make fixed-width bins based on given bin size.

Usage

```
fixedWidthBins(bamfile = NULL, assembly = NULL, chrom.lengths = NULL,
  chromosome.format, binsizes = 1e+06, chromosomes = NULL)
```

Arguments

`bamfile`	A BAM file from which the header is read to determine the chromosome lengths. If a `bamfile` is specified, option `assembly` is ignored.
`assembly`	An assembly from which the chromosome lengths are determined. Please see `fetchExtendedChromInfoFromUCSC` for available assemblies. This option is ignored if `bamfile` is specified. Alternatively a data.frame generated by `fetchExtendedChromInfoFromU`
`chrom.lengths`	A named character vector with chromosome lengths. Names correspond to chromosomes.
`chromosome.format`	A character specifying the format of the chromosomes if `assembly` is specified. Either 'NCBI' for (1,2,3 ...) or 'UCSC' for (chr1,chr2,chr3 ...). If a `bamfile` or `chrom.lengths` is supplied, the format will be chosen automatically.
`binsizes`	A vector of bin sizes in base pairs.
`chromosomes`	A subset of chromosomes for which the bins are generated.

Value

A `list()` of `GRanges` objects with fixed-width bins.

Author(s)

Aaron Taudt

Examples

```
## Make fixed-width bins of size 500kb and 1Mb
bins <- fixedWidthBins(assembly='mm10', chromosome.format='NCBI', binsizes=c(5e5,1e6))
bins
```

getSCEcoordinates *Get SCE coordinates*

Description

Extracts the coordinates of a sister chromatid exchanges (SCE) from an aneuBiHMM object.

Usage

```
getSCEcoordinates(model, resolution = c(3, 6), min.segwidth = 2,
  fragments = NULL, min.reads = 50)
```

Arguments

model	An aneuBiHMM object.
resolution	An integer vector specifying the resolution at bin level at which to scan for SCE events.
min.segwidth	Minimum segment length in bins when scanning for SCE events.
fragments	A GRanges object with read fragments or a file that contains such an object. These reads will be used for fine mapping of the SCE events.
min.reads	Minimum number of reads required for SCE refinement.

Value

A GRanges object containing the SCE coordinates.

Author(s)

Aaron Taudt

Examples

```
## Get an example BED file with single-cell-sequencing reads
bedfile <- system.file("extdata", "KK150311_VI_07.bam.bed.gz", package="AneuFinderData")
## Bin the BAM file into bin size 1Mp
binned <- binReads(bedfile, format='bed', assembly='hg19', binsize=1e6,
                   chromosomes=c(1:22,'X','Y'), pairedEndReads=TRUE)
## Fit the Hidden Markov Model
model <- findSCEs(binned[[1]], eps=0.1, max.time=60)
## Find sister chromatid exchanges
model$sce <- getSCEcoordinates(model)
print(model$sce)
plot(model)
```

getSegments *Extract segments and cluster*

Description

Extract segments and ID from a list of aneuHMM or aneuBiHMM objects and cluster if desired.

Usage

```
getSegments(hmms, cluster = TRUE, classes = NULL)
```

Arguments

hmms	A list of aneuHMM or aneuBiHMM objects or files that contain such objects.
cluster	Either TRUE or FALSE, indicating whether the samples should be clustered by similarity in their CNV-state.
classes	A vector with class labels the same length as hmms. If supplied, the clustering will be ordered optimally with respect to the class labels (see RearrangeJoseph).

Value

A list() with (clustered) segments and SCE coordinates.

heatmapAneuploidies *Plot aneuploidy state*

Description

Plot a heatmap of aneuploidy state for multiple samples. Samples can be clustered and the output can be returned as data.frame.

Usage

```
heatmapAneuploidies(hmms, ylabels = NULL, cluster = TRUE,
  as.data.frame = FALSE)
```

Arguments

hmms	A list of aneuHMM objects or files that contain such objects.
ylabels	A vector with labels for the y-axis. The vector must have the same length as hmms. If NULL the IDs from the aneuHMM objects will be used.
cluster	If TRUE, the samples will be clustered by similarity in their CNV-state.
as.data.frame	If TRUE, instead of a plot, a data.frame with the aneuploidy state for each sample will be returned.

Value

A ggplot object or a data.frame, depending on option as.data.frame.

Author(s)

Aaron Taudt

Examples

```
## Get results from a small-cell-lung-cancer
folder <- system.file("extdata", "primary-lung", "hmms", package="AneuFinderData")
files <- list.files(folder, full.names=TRUE)
## Plot the ploidy state per chromosome
heatmapAneuploidies(files, cluster=FALSE)
## Return the ploidy state as data.frame
df <- heatmapAneuploidies(files, cluster=FALSE, as.data.frame=TRUE)
head(df)
```

---

heatmapGenomewide *Genome wide heatmap of CNV-state*

Description

Plot a genome wide heatmap of copy number variation state. This heatmap is best plotted to file, because in most cases it will be too big for cleanly plotting it to screen.

Usage

```
heatmapGenomewide(hmms, ylabels = NULL, classes = NULL,
  classes.color = NULL, file = NULL, cluster = TRUE, plot.SCE = TRUE,
  hotspots = NULL)
```

Arguments

hmms	A list of aneuHMM objects or files that contain such objects.
ylabels	A vector with labels for the y-axis. The vector must have the same length as hmms. If NULL the IDs from the aneuHMM objects will be used.
classes	A character vector with the classification of the elements on the y-axis. The vector must have the same length as hmms. If specified the clustering algorithm will try to display similar categories together in the dendrogram.
classes.color	A (named) vector with colors that are used to distinguish classes. Names must correspond to the unique elements in classes.
file	A PDF file to which the heatmap will be plotted.
cluster	Either TRUE or FALSE, indicating whether the samples should be clustered by similarity in their CNV-state.
plot.SCE	Logical indicating whether SCE events should be plotted.
hotspots	A GRanges object with coordinates of genomic hotspots (see hotspotter).

Value

A ggplot object or NULL if a file was specified.

Examples

```
## Get results from a small-cell-lung-cancer
lung.folder <- system.file("extdata", "primary-lung", "hmms", package="AneuFinderData")
lung.files <- list.files(lung.folder, full.names=TRUE)
## Get results from the liver metastasis of the same patient
liver.folder <- system.file("extdata", "metastasis-liver", "hmms", package="AneuFinderData")
liver.files <- list.files(liver.folder, full.names=TRUE)
## Plot a clustered heatmap
classes <- c(rep('lung', length(lung.files)), rep('liver', length(liver.files)))
labels <- c(paste('lung',1:length(lung.files)), paste('liver',1:length(liver.files)))
heatmapGenomewide(c(lung.files, liver.files), ylabels=labels, classes=classes,
                  classes.color=c('blue','red'))
```

---

hotspotter *Find hotspots of genomic events*

Description

Find hotspots of genomic events by using kernel density estimation.

Usage

```
hotspotter(gr.list, bw, pval = 1e-08)
```

Arguments

gr.list	A list with GRanges object containing the coordinates of the genomic events.
bw	Bandwidth used for kernel density estimation (see density).
pval	P-value cutoff for hotspots.

Value

A GRanges object containing coordinates of hotspots with p-values.

Author(s)

Aaron Taudt

importBed *Read bed-file into GRanges*

Description

This is a simple convenience function to read a bed(.gz)-file into a GRanges object. The bed-file is expected to have the following fields: `chromosome, start, end, name, score, strand`.

Usage

```
importBed(bedfile, skip = 0, chromosome.format = "NCBI")
```

Arguments

`bedfile`	Filename of the bed or bed.gz file.
`skip`	Number of lines to skip at the beginning.
`chromosome.format`	Desired format of the chromosomes. Either 'NCBI' for (1,2,3 ...) or 'UCSC' for (chr1,chr2,chr3 ...).

Value

A GRanges object with the contents of the bed-file.

Author(s)

Aaron Taudt

Examples

```
## Get an example BED file with single-cell-sequencing reads
bedfile <- system.file("extdata", "KK150311_VI_07.bam.bed.gz", package="AneuFinderData")
## Import the file and skip the first 10 lines
data <- importBed(bedfile, skip=10)
```

initializeStates *Initialize state factor levels and distributions*

Description

Initialize the state factor levels and distributions for the specified states.

Usage

```
initializeStates(states)
```

Arguments

states	A subset of `c("zero-inflation","0-somy","1-somy","2-somy","3-somy","4-somy",...)`.

Value

A `list` with $labels, $distributions and $multiplicity values for the given states.

`karyotypeMeasures` *Measures for Karyotype Heterogeneity*

Description

Computes measures for karyotype heterogeneity. See the Details section for how these measures are defined.

Usage

```
karyotypeMeasures(hmms, normalChromosomeNumbers = NULL)
```

Arguments

hmms	A list with aneuHMM objects or a list of files that contain such objects.
normalChromosomeNumbers	A named integer vector with physiological copy numbers. This is useful to specify male and female samples, e.g. `c('X'=2)` for female samples and `c('X'=1,'Y'=1)` for male samples. The assumed default is '2' for all chromosomes.

Details

We define x as the vector of copy number states for each position. The number of HMMs is S. The measures are computed for each bin as follows:

Aneuploidy: $D = mean(abs(x - P))$, where P is the physiological number of chromosomes at that position.

Heterogeneity: $H = sum(table(x) * 0 : (length(table(x)) - 1))/S$

Value

A `list` with two `data.frames`, containing the karyotype measures $genomewide and $per.chromosome.

Author(s)

Aaron Taudt

Examples

```
## Get results from a small-cell-lung-cancer
lung.folder <- system.file("extdata", "primary-lung", "hmms", package="AneuFinderData")
lung.files <- list.files(lung.folder, full.names=TRUE)
## Get results from the liver metastasis of the same patient
liver.folder <- system.file("extdata", "metastasis-liver", "hmms", package="AneuFinderData")
liver.files <- list.files(liver.folder, full.names=TRUE)
normal.chrom.numbers <- rep(2, 23)
names(normal.chrom.numbers) <- c(1:22,'X')
lung <- karyotypeMeasures(lung.files, normalChromosomeNumbers=normal.chrom.numbers)
liver <- karyotypeMeasures(liver.files, normalChromosomeNumbers=normal.chrom.numbers)
print(lung$genomewide)
print(liver$genomewide)
```

---

loadGRangesFromFiles *Load GRanges from files*

Description

Load GRanges objects from file into a list.

Usage

```
loadGRangesFromFiles(files)
```

Arguments

files	A list of files that contain GRanges objects.

Value

A list() containing all loaded GRanges objects.

Author(s)

Aaron Taudt

Examples

```
## Not run:
## Get some files that you want to load
files <- list.files("folder-with-saved-GRanges-RData-files",
                    full.names=TRUE)
grlist <- loadGRangesFromFiles(files)
## Plot one of them
plot(grlist[[1]]
## End(Not run)
```

loadHmmsFromFiles *Load HMMs from files*

Description

Load aneuHMM objects from file into a list.

Usage

```
loadHmmsFromFiles(hmms, strict = TRUE)
```

Arguments

hmms	A list of files that contain aneuHMM objects.
strict	If any of the loaded objects is not a aneuHMM object, an error (strict=TRUE) or a warning (strict=FALSE) will be generated.

Value

A list() containing all loaded aneuHMM objects.

Author(s)

Aaron Taudt

Examples

```
## Get some files that you want to load
folder <- system.file("extdata", "primary-lung", "hmms", package="AneuFinderData")
files <- list.files(folder, full.names=TRUE)
## Load and plot the first then
hmms <- loadHmmsFromFiles(files[1:10])
lapply(hmms, plot, type='profile')
```

plot.aneuBiHMM *Plotting function for aneuBiHMM objects*

Description

Make different types of plots for aneuBiHMM objects.

Usage

```
## S3 method for class 'aneuBiHMM'
plot(x, type = "profile", ...)
```

Arguments

x	An aneuBiHMM object.
type	Type of the plot, one of c('profile', 'histogram', 'karyogram'). You can also specify the type with an integer number. profile An profile with read counts and CNV-state. histogram A histogram of binned read counts with fitted mixture distribution. karyogram A karyogram-like chromosome overview with CNV-state.
...	Additional arguments for the different plot types.

Value

A ggplot object.

---

plot.aneuHMM *Plotting function for aneuHMM objects*

Description

Make different types of plots for aneuHMM objects.

Usage

```
## S3 method for class 'aneuHMM'
plot(x, type = "profile", ...)
```

Arguments

x	An aneuHMM object.
type	Type of the plot, one of c('profile', 'histogram', 'karyogram'). You can also specify the type with an integer number. karyogram A karyogram-like chromosome overview with CNV-state. histogram A histogram of binned read counts with fitted mixture distribution. karyogram An profile with read counts and CNV-state.
...	Additional arguments for the different plot types.

Value

A ggplot object.

plot.character *Plotting function for saved* **AneuFinder** *objects*

Description

Convenience function that loads and plots a **AneuFinder** object in one step.

Usage

```
## S3 method for class 'character'
plot(x, ...)
```

Arguments

x	A filename that contains either binned.data or a aneuHMM.
...	Additional arguments.

Value

A ggplot object.

plot.GRanges *Plotting function for binned read counts*

Description

Make plots for binned read counts from binned.data.

Usage

```
## S3 method for class 'GRanges'
plot(x, type = "profile", ...)
```

Arguments

x	A GRanges object with binned read counts.
type	Type of the plot, one of c('profile', 'histogram', 'karyogram'). You can also specify the type with an integer number. karyogram A karyogram-like chromosome overview with read counts. histogram A histogram of read counts. profile An profile with read counts.
...	Additional arguments for the different plot types.

Value

A ggplot object.

plotBinnedDataHistogram *Plot a histogram of binned read counts*

Description

Plot a histogram of binned read counts from binned.data

Usage

```
plotBinnedDataHistogram(binned.data, strand = "*", chromosome = NULL,
  start = NULL, end = NULL)
```

Arguments

binned.data	A GRanges object containing binned read counts in meta-column 'counts'.
strand	One of c('+','-','*'). Plot the histogram only for the specified strand.
chromosome, start, end	Plot the histogram only for the specified chromosome, start and end position.

Value

A ggplot object.

plotKaryogram *Karyogram-like chromosome overview*

Description

Plot a karyogram-like chromosome overview with read counts and CNV-state from a aneuHMM object or binned.data.

Usage

```
plotKaryogram(model, both.strands = FALSE, plot.SCE = FALSE, file = NULL)
```

Arguments

model	A aneuHMM object or binned.data.
both.strands	If TRUE, strands will be plotted separately.
plot.SCE	Logical indicating whether SCE events should be plotted.
file	A PDF file where the plot will be saved.

Value

A ggplot object or NULL if a file was specified.

`plotProfile` *Read count and CNV profile*

Description

Plot a profile with read counts and CNV-state from a `aneuHMM` object or `binned.data`.

Usage

```
plotProfile(model, both.strands = FALSE, plot.SCE = TRUE, file = NULL)
```

Arguments

`model`	A `aneuHMM` object or `binned.data`.
`both.strands`	If `TRUE`, strands will be plotted separately.
`plot.SCE`	Logical indicating whether SCE events should be plotted.
`file`	A PDF file where the plot will be saved.

Value

A `ggplot` object or `NULL` if a file was specified.

`plotUnivariateHistogram`
Plot a histogram of binned read counts with fitted mixture distribution

---

Description

Plot a histogram of binned read counts from with fitted mixture distributions from a `aneuHMM` object.

Usage

```
plotUnivariateHistogram(model, state = NULL, strand = "*",
  chromosome = NULL, start = NULL, end = NULL)
```

Arguments

`model`	A `aneuHMM` object.
`state`	Plot the histogram only for the specified CNV-state.
`strand`	One of c('+','-','*'). Plot the histogram only for the specified strand.
`chromosome, start, end`	Plot the histogram only for the specified chromosome, start and end position.

Value

A `ggplot` object.

qualityControl *Quality control measures for binned read counts*

Description

Calculate various quality control measures on binned read counts.

Usage

```
qc.spikiness(counts)

qc.entropy(counts)

qc.bhattacharyya(hmm)
```

Arguments

counts	A vector of binned read counts.
hmm	An aneuHMM object.

Details

The Shannon entropy is defined as $S = -sum(n * log(n))$, where $n = counts/sum(counts)$.

Spikyness is defined as $K = sum(abs(diff(counts)))/sum(counts)$.

Value

A numeric.

Functions

- qc.spikiness: Calculate the spikiness of a library
- qc.entropy: Calculate the Shannon entropy of a library
- qc.bhattacharyya: Calculate the Bhattacharyya distance between the '1-somy' and '2-somy' distribution

Author(s)

Aaron Taudt

readConfig *Read AneuFinder configuration file*

Description

Read an AneuFinder configuration file into a list structure. The configuration file has to be specified in INI format. R expressions can be used and will be evaluated.

Usage

```
readConfig(configfile)
```

Arguments

configfile	Path to the configuration file

Value

A list with one entry for each element in configfile.

Author(s)

Aaron Taudt

simulateReads *Simulate reads from genome*

Description

Simulate single or paired end reads from any **BSgenome** object. These simulated reads can be mapped to the reference genome using any aligner to produce BAM files that can be used for mappability correction.

Usage

```
simulateReads(bsgenome, readLength, bamfile, file,
  pairedEndFragmentLength = NULL, every.X.bp = 500)
```

Arguments

bsgenome	A **BSgenome** object containing the sequence of the reference genome.
readLength	The length in base pairs of the simulated reads that are written to file.
bamfile	A BAM file. This file is used to estimate the distribution of Phred quality scores.
file	The filename that is written to disk. The ending .fastq.gz will be appended.
pairedEndFragmentLength	If this option is specified, paired end reads with length readLength will be simulated coming from both ends of fragments of this size. NOT IMPLEMENTED YET.
every.X.bp	Stepsize for simulating reads. A read fragment will be simulated every X bp.

Details

Reads are simulated by splitting the genome into reads with the specified readLength.

Value

A fastq.gz file is written to disk.

Author(s)

Aaron Taudt

Examples

```
## Get an example BAM file with single-cell-sequencing reads
bamfile <- system.file("extdata", "BB150803_IV_074.bam", package="AneuFinderData")
## Simulate 51bp reads for at a distance of every 5000bp
if (require(BSgenome.Mmusculus.UCSC.mm10)) {
simulateReads(BSgenome.Mmusculus.UCSC.mm10, bamfile=bamfile, readLength=51,
              file=tempfile(), every.X.bp=5000)
}
```

---

subsetByCNVprofile *Get IDs of a subset of models*

Description

Get the IDs of models that have a certain CNV profile. The result will be TRUE for models that overlap all specified ranges in profile by at least one base pair with the correct state.

Usage

```
subsetByCNVprofile(hmms, profile)
```

Arguments

hmms	A list of aneuHMM objects or files that contain such objects.
profile	A GRanges object with metadata column 'expected.state' and optionally columns 'expected.mstate' and 'expected.pstate'.

Value

A named logical vector with TRUE for all models that are concordant with the given profile.

Examples

```
## Get results from a small-cell-lung-cancer
lung.folder <- system.file("extdata", "primary-lung", "hmms", package="AneuFinderData")
lung.files <- list.files(lung.folder, full.names=TRUE)
## Get all files that have a 3-somy on chromosome 1 and 4-somy on chromosome 2
profile <- GRanges(seqnames=c('1','2'), ranges=IRanges(start=c(1,1), end=c(195471971,182113224)),
                   expected.state=c('3-somy','4-somy'))
ids <- subsetByCNVprofile(lung.files, profile)
print(which(ids))
```

---

transCoord *Transform genomic coordinates*

Description

Add two columns with transformed genomic coordinates to the GRanges object. This is useful for making genomewide plots.

Usage

```
transCoord(gr)
```

Arguments

gr	A GRanges object.

Value

The input GRanges with two additional metadata columns 'start.genome' and 'end.genome'.

`univariate.findCNVs` *Find copy number variations (univariate)*

Description

`findCNVs` classifies the binned read counts into several states which represent copy-number-variation.

Usage

```
univariate.findCNVs(binned.data, ID = NULL, eps = 0.1, init = "standard",
  max.time = -1, max.iter = -1, num.trials = 1, eps.try = NULL,
  num.threads = 1, count.cutoff.quantile = 0.999, strand = "*",
  states = c("zero-inflation", paste0(0:10, "-somy")),
  most.frequent.state = "2-somy", algorithm = "EM", initial.params = NULL)
```

Arguments

`binned.data`	A GRanges object with binned read counts.
`ID`	An identifier that will be used to identify this sample in various downstream functions. Could be the file name of the `binned.data` for example.
`eps`	Convergence threshold for the Baum-Welch algorithm.
`init`	One of the following initialization procedures: `standard` The negative binomial of state '2-somy' will be initialized with `mean=mean(counts)`, `var=var(counts)`. This procedure usually gives good convergence. `random` Mean and variance of the negative binomial of state '2-somy' will be initialized with random values (in certain boundaries, see source code). Try this if the `standard` procedure fails to produce a good fit.
`max.time`	The maximum running time in seconds for the Baum-Welch algorithm. If this time is reached, the Baum-Welch will terminate after the current iteration finishes. The default -1 is no limit.
`max.iter`	The maximum number of iterations for the Baum-Welch algorithm. The default -1 is no limit.
`num.trials`	The number of trials to find a fit where state `most.frequent.state` is most frequent. Each time, the HMM is seeded with different random initial values.
`eps.try`	If code num.trials is set to greater than 1, `eps.try` is used for the trial runs. If unset, `eps` is used.
`num.threads`	Number of threads to use. Setting this to >1 may give increased performance.
`count.cutoff.quantile`	A quantile between 0 and 1. Should be near 1. Read counts above this quantile will be set to the read count specified by this quantile. Filtering very high read counts increases the performance of the Baum-Welch fitting procedure. However, if your data contains very few peaks they might be filtered out. Set `count.cutoff.quantile=1` in this case.
`strand`	Run the HMM only for the specified strand. One of `c('+', '-', '*')`.

states	A subset or all of c("zero-inflation","0-somy","1-somy","2-somy","3-somy","4-somy",...). This vector defines the states that are used in the Hidden Markov Model. The order of the entries must not be changed.
most.frequent.state	One of the states that were given in states or 'none'. The specified state is assumed to be the most frequent one. This can help the fitting procedure to converge into the correct fit.
algorithm	One of c('baumWelch','EM'). The expectation maximization ('EM') will find the most likely states and fit the best parameters to the data, the 'baumWelch' will find the most likely states using the initial parameters.
initial.params	A aneuHMM object or file containing such an object from which initial starting parameters will be extracted.

Value

An aneuHMM object.

`variableWidthBins` *Make variable-width bins*

Description

Make variable-width bins based on a reference BAM file. This can be a simulated file (produced by simulateReads and aligned with your favourite aligner) or a real reference.

Usage

```
variableWidthBins(reads, binsizes, chromosomes = NULL)
```

Arguments

reads	A GRanges with reads. See bam2GRanges and bed2GRanges.
binsizes	A vector with binsizes. Resulting bins will be close to the specified binsizes.
chromosomes	A subset of chromosomes for which the bins are generated.

Details

Variable-width bins are produced by first binning the reference BAM file with fixed-width bins and selecting the desired number of reads per bin as the (non-zero) maximum of the histogram. A new set of bins is then generated such that every bin contains the desired number of reads.

Value

A list() of GRanges objects with variable-width bins.

Author(s)

Aaron Taudt

Examples

```
## Get an example BED file with single-cell-sequencing reads
bedfile <- system.file("extdata", "KK150311_VI_07.bam.bed.gz", package="AneuFinderData")
## Read the file into a GRanges object
reads <- bed2GRanges(bedfile, assembly='mm10', chromosomes=c(1:19,'X','Y'),
                     min.mapq=10, remove.duplicate.reads=TRUE)
## Make variable-width bins of size 500kb and 1Mb
bins <- variableWidthBins(reads, binsizes=c(5e5,1e6))
## Plot the distribution of binsizes
hist(width(bins[['1e+06']]), breaks=50)
```

---

writeConfig *Write AneuFinder configuration file*

Description

Write an AneuFinder configuration file from a list structure.

Usage

```
writeConfig(conf, configfile)
```

Arguments

conf	A list structure with parameter values. Each entry will be written in one line.
configfile	Filename of the outputfile.

Value

NULL

Author(s)

Aaron Taudt

zinbinom *The Zero-inflated Negative Binomial Distribution*

Description

Density, distribution function, quantile function and random generation for the zero-inflated negative binomial distribution with parameters `w`, `size` and `prob`.

Usage

```
dzinbinom(x, w, size, prob, mu)

pzinbinom(q, w, size, prob, mu, lower.tail = TRUE)

qzinbinom(p, w, size, prob, mu, lower.tail = TRUE)

rzinbinom(n, w, size, prob, mu)
```

Arguments

`x`	Vector of (non-negative integer) quantiles.
`w`	Weight of the zero-inflation. `0 <= w <= 1`.
`size`	Target for number of successful trials, or dispersion parameter (the shape parameter of the gamma mixing distribution). Must be strictly positive, need not be integer.
`prob`	Probability of success in each trial. `0 < prob <= 1`.
`mu`	Alternative parametrization via mean: see 'Details'.
`q`	Vector of quantiles.
`lower.tail`	logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.
`p`	Vector of probabilities.
`n`	number of observations. If `length(n) > 1`, the length is taken to be the number required.

Details

The zero-inflated negative binomial distribution with `size` $= n$ and `prob` $= p$ has density

$$p(x) = w + (1 - w)\frac{\Gamma(x + n)}{\Gamma(n)x!}p^n(1 - p)^x$$

for $x = 0$, $n > 0$, $0 < p \leq 1$ and $0 \leq w \leq 1$.

$$p(x) = (1 - w)\frac{\Gamma(x + n)}{\Gamma(n)x!}p^n(1 - p)^x$$

for $x = 1, 2, \ldots$, $n > 0$, $0 < p \leq 1$ and $0 \leq w \leq 1$.

Value

dzinbinom gives the density, pzinbinom gives the distribution function, qzinbinom gives the quantile function, and rzinbinom generates random deviates.

Functions

- `dzinbinom`: gives the density
- `pzinbinom`: gives the cumulative distribution function
- `qzinbinom`: gives the quantile function
- `rzinbinom`: random number generation

Author(s)

Matthias Heinig, Aaron Taudt

See Also

Distributions for standard distributions, including `dbinom` for the binomial, `dnbinom` for the negative binomial, `dpois` for the Poisson and `dgeom` for the geometric distribution, which is a special case of the negative binomial.

Index