Package 'AneuFinder'

November 4, 2024

Type Package

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

Version 1.35.0

Author Aaron Taudt, Bjorn Bakker, David Porubsky

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

Description AneuFinder implements functions for copy-number detection, breakpoint detection, and karyotype and heterogeneity analysis in single-cell whole genome sequencing and strand-seq data.

Depends R (>= 3.5), GenomicRanges, ggplot2, cowplot, AneuFinderData

Imports methods, utils, grDevices, graphics, stats, foreach, doParallel, BiocGenerics (>= 0.31.6), S4Vectors, GenomeInfoDb, IRanges, Rsamtools, bamsignals, DNAcopy, ecp, Biostrings, GenomicAlignments, reshape2, ggdendro, ggrepel, mclust

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

License Artistic-2.0

LazyLoad yes

VignetteBuilder knitr

biocViews ImmunoOncology, Software, Sequencing, SingleCell, CopyNumberVariation, GenomicVariation, HiddenMarkovModel, WholeGenome

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

RoxygenNote 6.0.1

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

git_branch devel

git_last_commit 42b7837

git_last_commit_date 2024-10-29

Repository Bioconductor 3.21

Date/Publication 2024-11-04

Contents

`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 findCNVs.strandseq 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-class object containing the genomic bin coordinates, their read count and state classification.
segments	A GRanges-class 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.strandseq

Aneufinder *Wrapper function for the* AneuFinder *package*

Description

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

Usage

```
Aneufinder(inputfolder, outputfolder, configfile = NULL, numCPU = 1,
  reuse.existing.files = TRUE, binsizes = 1e+06, stepsizes = binsizes,
  variable.width.reference = NULL, reads.per.bin = NULL,
  pairedEndReads = FALSE, assembly = NULL, chromosomes = NULL,
  remove.duplicate.reads = TRUE, min.mapq = 10, blacklist = NULL,
  use.bamsignals = FALSE, reads.store = FALSE, correction.method = NULL,
  GC.BSgenome = NULL, method = c("edivisive"), strandseq = FALSE,
  R = 10, sig.lvl = 0.1, eps = 0.01, max.time = 60, max.iter = 5000,
  num.trials = 15, states = c("zero-inflation", paste0(0:10, "-somy")),
  confint = NULL, refine.breakpoints = FALSE, hotspot.bandwidth = NULL,
  hotspot.pval = 0.05, 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.
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.
stepsizes	A vector of step sizes the same length as binsizes. Only used for method="HMM".
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).

assembly — Please see getChromInfoFromUCSC for available assemblies. Only necessary when importing BED files. BAM files are handled automatically. Alternatively a data.frame with columns 'chromosome' and 'length'.

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=NA to keep all reads.

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

use.bamsignals — If TRUE the **bamsignals** package will be used for binning. This gives a tremendous performance increase for the binning step. reads.store and calc.complexity will be set to FALSE in this case.

reads.store — Set reads.store=TRUE to store read fragments as RData in folder 'data' and as BED files in 'BROWSERFILES/data'. This option will force use.bamsignals=FALSE.

correction.method — Correction methods to be used for the binned read counts. Currently only 'GC'.

GC.BSgenome — A BSgenome object which contains the DNA sequence that is used for the GC correction.

method — Any combination of c('HMM','dnacopy','edivisive'). Option method='HMM' uses a Hidden Markov Model as described in doi:10.1186/s13059-016-0971-7 to call copy numbers. Option 'dnacopy' uses segment from the **DNAcopy** package to call copy numbers similarly to the method proposed in doi:10.1038/nmeth.3578, which gives more robust but less sensitive results compared to the HMM. Option 'edivisive' (DEFAULT) works like option 'dnacopy' but uses the e.divisive function from the **ecp** package for segmentation.

strandseq — A logical indicating whether the data comes from Strand-seq experiments. If TRUE, both strands carry information and are treated separately.

R — method-edivisive: The maximum number of random permutations to use in each iteration of the permutation test (see e.divisive). Increase this value to increase accuracy on the cost of speed.

sig.lvl — method-edivisive: The level at which to sequentially test if a proposed change point is statistically significant (see e.divisive). Increase this value to find more breakpoints.

eps — method-HMM: Convergence threshold for the Baum-Welch algorithm.

max.time — method-HMM: 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. Set max.time = -1 for no limit.

max.iter — method-HMM: The maximum number of iterations for the Baum-Welch algorithm. Set max.iter = -1 for no limit.

num.trials — method-HMM: 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	method-HMM: A subset or all of c("zero-inflation","0-somy","1-somy","2-somy","3-somy","4- This vector defines the states that are used in the Hidden Markov Model. The order of the entries must not be changed.
confint	Desired confidence interval for breakpoints. Set confint=NULL to disable confidence interval estimation. Confidence interval estimation will force reads.store=TRUE.
refine.breakpoints	A logical indicating whether breakpoints from the HMM should be refined with read-level information. refine.breakpoints=TRUE will force reads.store=TRUE.
hotspot.bandwidth	A vector the same length as binsizes with bandwidths for breakpoint hotspot detection (see hotspotter for further details). If NULL, the bandwidth will be chosen automatically as the average distance between reads.
hotspot.pval	P-value for breakpoint hotspot detection (see hotspotter for further details). Set hotspot.pval = NULL to skip hotspot detection.
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")
## 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-class object containing the genomic bin coordinates, their read count and state classification.

`segments` A GRanges-class 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

`annotateBreakpoints` *Annotate breakpoints*

Description

Annotate breakpoints as sister-chromatid-exchange (SCE), copy-number-breakpoint (CNB).

Usage

```
annotateBreakpoints(breakpoints)
```

Arguments

`breakpoints` A `GRanges-class` as returned by `getBreakpoints`.

Value

The input GRanges-class with additinal column 'type'.

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 data into bin size 1Mp
readfragments <- binReads(bedfile, assembly='mm10', binsize=1e6,
                  chromosomes=c(1:19,'X','Y'), reads.return=TRUE)
binned <- binReads(bedfile, assembly='mm10', binsize=1e6,
                  chromosomes=c(1:19,'X','Y'))
## Fit the Hidden Markov Model
model <- findCNVs.strandseq(binned[[1]])
## Add confidence intervals
breakpoints <- getBreakpoints(model, readfragments)
```

---

bam2GRanges *Import BAM file into GRanges*

Description

Import aligned reads from a BAM file into a GRanges-class 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=NA 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-class 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. Uses the Rsamtools::scanBamWhat function. See Rsamtools::ScanBamParam to see what is available.

Value

A GRanges-class 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-class 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 getChromInfoFromUCSC for available assemblies. Only necessary when importing BED files. BAM files are handled automatically. Alternatively a data.frame with columns 'chromosome' and 'length'.
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=NA to keep all reads.
max.fragment.width	Maximum allowed fragment length. This is to filter out erroneously wrong fragments.
blacklist	A GRanges-class or a bed(.gz) file with blacklisted regions. Reads falling into those regions will be discarded.

Value

A GRanges-class 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)
```

---

bi.edivisive.findCNVs *Find copy number variations (edivisive, bivariate)*

Description

Classify the binned read counts into several states which represent copy-number-variation. The function uses the e.divisive function to segment the genome.

Usage

```
bi.edivisive.findCNVs(binned.data, ID = NULL, CNgrid.start = 0.5, R = 10,
  sig.lvl = 0.1)
```

Arguments

binned.data	A GRanges-class 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.
CNgrid.start	Start parameter for the CNgrid variable. Very empiric. Set to 1.5 for normal data and 0.5 for Strand-seq data.
R	method-edivisive: The maximum number of random permutations to use in each iteration of the permutation test (see e.divisive). Increase this value to increase accuracy on the cost of speed.
sig.lvl	method-edivisive: The level at which to sequentially test if a proposed change point is statistically significant (see e.divisive). Increase this value to find more breakpoints.

Value

An aneuHMM object.

biDNAcopy.findCNVs *Find copy number variations (DNAcopy, bivariate)*

Description

biDNAcopy.findCNVs classifies the binned read counts into several states which represent copy-number-variation using read count information from both strands.

Usage

```
biDNAcopy.findCNVs(binned.data, ID = NULL, CNgrid.start = 0.5)
```

Arguments

binned.data	A GRanges-class 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.
CNgrid.start	Start parameter for the CNgrid variable. Very empiric. Set to 1.5 for normal data and 0.5 for Strand-seq data.

Value

An aneuHMM object.

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

Description

biHMM.findCNVs finds CNVs using read count information from both strands.

Usage

```
biHMM.findCNVs(binned.data, ID = NULL, eps = 0.01, 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,
  verbosity = 1)
```

Arguments

binned.data	A GRanges-class object with binned read counts. Alternatively a GRangesList object with offsetted 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	method-HMM: Convergence threshold for the Baum-Welch algorithm.
init	method-HMM: 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	method-HMM: 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. Set max.time = -1 for no limit.
max.iter	method-HMM: The maximum number of iterations for the Baum-Welch algorithm. Set max.iter = -1 for no limit.
num.trials	method-HMM: 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	method-HMM: 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	method-HMM: Number of threads to use. Setting this to >1 may give increased performance.
count.cutoff.quantile	method-HMM: 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	method-HMM: A subset or all of c("zero-inflation","0-somy","1-somy","2-somy","3-somy","4- 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	method-HMM: One of the states that were given in states. The specified state is assumed to be the most frequent one. This can help the fitting procedure to converge into the correct fit.
algorithm	method-HMM: 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	method-HMM: A aneuHMM object or file containing such an object from which initial starting parameters will be extracted.
verbosity	method-HMM: Integer specifying the verbosity of printed messages.

Value

An aneuBiHMM object.

`binned.data` *Binned read counts*

Description

A GRanges-class 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, 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,
  stepsizes = NULL, reads.per.bin = NULL, reads.per.step = NULL,
  bins = NULL, variable.width.reference = 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, use.bamsignals = FALSE)
```

Arguments

file	A file with aligned reads. Alternatively a GRanges-class with aligned reads.
assembly	Please see getChromInfoFromUCSC for available assemblies. Only necessary when importing BED files. BAM files are handled automatically. Alternatively a data.frame with columns 'chromosome' and 'length'.
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=NA 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-class 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.
stepsizes	A vector of step sizes the same length as binsizes. Only used for method="HMM".
reads.per.bin	Approximate number of desired reads per bin. The bin size will be selected accordingly. Output files are produced for each value.
reads.per.step	Approximate number of desired reads per step.
bins	A named list with GRanges-class 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.
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. Will be ignored if `use.bamsignals=TRUE`.

`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-class 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.

`use.bamsignals` If TRUE the **bamsignals** package will be used for binning. This gives a tremendous performance increase for the binning step. `reads.store` and `calc.complexity` will be set to FALSE in this case.

Details

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

Value

The function produces a `list()` of GRanges-class or GRangesList objects with meta data columns 'counts', 'mcounts', 'pcounts' that contain the total, minus and plus 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, assembly='mm10', binsize=1e6,
                   chromosomes=c(1:19,'X','Y'))
print(binned)
```

`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 be filtered with a sensible cutoff. See the examples section for details.

Usage

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

Arguments

files	A character vector of either BAM or BED files.
assembly	Please see getChromInfoFromUCSC for available assemblies. Only necessary when importing BED files. BAM files are handled automatically. Alternatively a data.frame with columns 'chromosome' and 'length'.
bins	A list with one GRanges-class with binned read counts generated by fixedWidthBins.
min.mapq	Minimum mapping quality when importing from BAM files. Set min.mapq=NA 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-class 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, 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",
  "complexity", "sos"), orderBy = "spikiness", reverseOrder = FALSE)
```

Arguments

hmms	A list of aneuHMM objects or a character vector with 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','
orderBy	The quality measure to order the clusters by. Default is 'spikiness'.
reverseOrder	Logical indicating whether the ordering by orderBy is reversed.

Details

Please see getQC for a brief description of the quality measures.

Value

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

Author(s)

Aaron Taudt

See Also

getQC

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])
```

clusterHMMs *Cluster objects*

Description

Cluster a list of aneuHMM or aneuBiHMM objects by similarity in their CNV-state.

Usage

```
clusterHMMs(hmms, cluster = TRUE, exclude.regions = NULL)
```

Arguments

hmms	A list of aneuHMM or aneuBiHMM objects or a character vector of files that contains such objects.
cluster	Either TRUE or FALSE, indicating whether the samples should be clustered by similarity in their CNV-state.
exclude.regions	A GRanges-class with regions that will be excluded from the computation of the clustering. This can be useful to exclude regions with artifacts.

Value

An list() with ordered ID indices and the hierarchical clustering.

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)
models <- loadFromFiles(lung.files)
## Not run:
# Plot unclustered heatmap
heatmapGenomewide(models, cluster=FALSE)
## End(Not run)
## Cluster and reorder the models
clust <- clusterHMMs(models)
models <- models[clust$IDorder]
## Not run:
# Plot re-ordered heatmap
heatmapGenomewide(models, cluster=FALSE)
## End(Not run)
```

`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, 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("+", "-"))

breakpointColors(breaktypes = c("CNB", "SCE", "CNB+SCE", "other"))
```

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('+','-','*').
breaktypes	A character vector with breakpoint types whose color should be returned. Any combination of c('CNB','SCE','CNB+SCE','other').

Value

A character vector with colors.

Functions

- stateColors: Colors that are used for the states.
- strandColors: Colors that are used to distinguish strands.
- breakpointColors: Colors that are used for breakpoint types.

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)

## Make a nice pie chart with the AneuFinder breakpoint-type color scheme
breakpointcolors <- breakpointColors()
pie(rep(1,length(breakpointcolors)), labels=names(breakpointcolors), col=breakpointcolors)
```

---

compareMethods *Compare copy number calling methods*

Description

Compare two sets of aneuHMM objects generated by different methods (see option method of findCNVs).

Usage

```
compareMethods(models1, models2)
```

Arguments

models1	A list of aneuHMM objects or a character vector with files that contain such objects.
models2	A list of aneuHMM objects or a character vector with files that contain such objects. IDs of the models must match the ones in models1.

Value

A data.frame with one column 'concordance' which gives the fraction of the genome that is called concordantly between both models.

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)
## Compare the models with themselves (non-sensical)
df <- compareMethods(files, files)
head(df)
```

`compareModels` *Compare copy number models*

Description

Compare two aneuHMM objects. The function computes the fraction of copy number calls that is concordant between both models.

Usage

```
compareModels(model1, model2)
```

Arguments

`model1`	An aneuHMM object or file that contains such an object.
`model2`	An aneuHMM object or file that contains such an object.

Value

A numeric.

Author(s)

Aaron Taudt

consensusSegments *Make consensus segments*

Description

Make consensus segments from a list of aneuHMM or aneuBiHMM objects.

Usage

```
consensusSegments(hmms)
```

Arguments

hmms	A list of aneuHMM or aneuBiHMM objects or a character vector of files that contains such objects.

Details

The function will produce a GRanges-class object using the GenomicRanges::disjoin function on all extracted $segment entries.

Value

A GRanges-class.

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 consensus segments and states
consensusSegments(lung.files)
```

correctGC *GC correction*

Description

Correct a list of binned.data by GC content.

Usage

```
correctGC(binned.data.list, GC.BSgenome, same.binsize = FALSE,
  method = "loess", return.plot = FALSE, bins = NULL)
```

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 and stepsize.
method	One of c('quadratic', 'loess'). Option method='quadratic' uses the method described in the Supplementary of citation("AneuFinder"). Option method='loess' uses a loess fit to adjust the read count.
return.plot	Set to TRUE if plots should be returned for visual assessment of the GC correction.
bins	A binned.data object with meta-data column 'GC'. If this is specified, GC.BSgenome is ignored. Beware, no format checking is done.

Details

Two methods are available for GC correction: Option method='quadratic' uses the method described in the Supplementary of citation("AneuFinder"). Option method='loess' uses a loess fit to adjust the read count.

Value

A list() with binned.data objects with adjusted read counts. Alternatively a list() with ggplot objects if return.plot=TRUE.

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, 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)
}
```

DNAcopy.findCNVs *Find copy number variations (DNAcopy, univariate)*

Description

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

Usage

```
DNAcopy.findCNVs(binned.data, ID = NULL, CNgrid.start = 1.5, strand = "*")
```

Arguments

binned.data	A GRanges-class 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.
CNgrid.start	Start parameter for the CNgrid variable. Very empiric. Set to 1.5 for normal data and 0.5 for Strand-seq data.
strand	Find copy-numbers only for the specified strand. One of c('+', '-', '*').

Value

An aneuHMM object.

edivisive.findCNVs *Find copy number variations (edivisive, univariate)*

Description

Classify the binned read counts into several states which represent copy-number-variation. The function uses the e.divisive function to segment the genome.

Usage

```
edivisive.findCNVs(binned.data, ID = NULL, CNgrid.start = 1.5,
  strand = "*", R = 10, sig.lvl = 0.1)
```

Arguments

binned.data	A GRanges-class 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.
CNgrid.start	Start parameter for the CNgrid variable. Very empiric. Set to 1.5 for normal data and 0.5 for Strand-seq data.
strand	Find copy-numbers only for the specified strand. One of c('+', '-', '*').
R	method-edivisive: The maximum number of random permutations to use in each iteration of the permutation test (see e.divisive). Increase this value to increase accuracy on the cost of speed.
sig.lvl	method-edivisive: The level at which to sequentially test if a proposed change point is statistically significant (see e.divisive). Increase this value to find more breakpoints.

Value

An aneuHMM object.

---

estimateComplexity *Estimate library complexity*

Description

Estimate library complexity using a very simple "Michaelis-Menten" approach.

Usage

```
estimateComplexity(reads)
```

Arguments

reads	A GRanges-class 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, trackname = NULL, cluster = TRUE,
  export.CNV = TRUE, export.breakpoints = TRUE)

exportReadCounts(hmms, filename)

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

Arguments

hmms	A list of aneuHMM objects or a character vector with 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.
trackname	The name that will be used as track name and description in the header.
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.breakpoints	A logical, indicating whether breakpoints shall be exported.
gr	A GRanges-class object.
header	A logical indicating whether the output file will have a heading track line (TRUE) or not (FALSE).
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
thickStart, thickEnd	A vector of the same length as gr, which will be used for the 'thickStart' and 'thickEnd' columns in the BED file.

as.wiggle	A logical indicating whether a variableStep-wiggle file will be exported instead of a BED file. If TRUE, wiggle.value must be specified.
wiggle.val	A vector of the same length as gr, which will be used for the values in the wiggle file.

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-class 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-class 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-class 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 <- loadFromFiles(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-numbers.

Usage

```
findCNVs(binned.data, ID = NULL, method = "edivisive", strand = "*",
  R = 10, sig.lvl = 0.1, eps = 0.01, init = "standard", max.time = -1,
  max.iter = 1000, num.trials = 15, eps.try = max(10 * eps, 1),
  num.threads = 1, count.cutoff.quantile = 0.999,
  states = c("zero-inflation", paste0(0:10, "-somy")),
  most.frequent.state = "2-somy", algorithm = "EM", initial.params = NULL,
  verbosity = 1)
```

Arguments

binned.data	A GRanges-class 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.

`method` Any combination of `c('HMM','dnacopy','edivisive')`. Option `method='HMM'` uses a Hidden Markov Model as described in doi:10.1186/s13059-016-0971-7 to call copy numbers. Option `'dnacopy'` uses `segment` from the **DNAcopy** package to call copy numbers similarly to the method proposed in doi:10.1038/nmeth.3578, which gives more robust but less sensitive results compared to the HMM. Option `'edivisive'` (DEFAULT) works like option `'dnacopy'` but uses the `e.divisive` function from the **ecp** package for segmentation.

`strand` Find copy-numbers only for the specified strand. One of `c('+', '-', '*')`.

`R` method-edivisive: The maximum number of random permutations to use in each iteration of the permutation test (see `e.divisive`). Increase this value to increase accuracy on the cost of speed.

`sig.lvl` method-edivisive: The level at which to sequentially test if a proposed change point is statistically significant (see `e.divisive`). Increase this value to find more breakpoints.

`eps` method-HMM: Convergence threshold for the Baum-Welch algorithm.

`init` method-HMM: 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` method-HMM: 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. Set `max.time = -1` for no limit.

`max.iter` method-HMM: The maximum number of iterations for the Baum-Welch algorithm. Set `max.iter = -1` for no limit.

`num.trials` method-HMM: 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` method-HMM: 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` method-HMM: Number of threads to use. Setting this to >1 may give increased performance.

`count.cutoff.quantile`
method-HMM: 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` method-HMM: A subset or all of `c("zero-inflation","0-somy","1-somy","2-somy","3-somy","4-`
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`
method-HMM: One of the states that were given in `states`. The specified state is assumed to be the most frequent one. This can help the fitting procedure to converge into the correct fit.

`algorithm`	method-HMM: 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`	method-HMM: A aneuHMM object or file containing such an object from which initial starting parameters will be extracted.
`verbosity`	method-HMM: Integer specifying the verbosity of printed messages.

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 data into bin size 1Mp
binned <- binReads(bedfile, assembly='mm10', binsize=1e6,
                   chromosomes=c(1:19,'X','Y'))
## Find copy-numbers
model <- findCNVs(binned[[1]])
## Check the fit
plot(model, type='histogram')
```

---

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

Description

`findCNVs.strandseq` classifies the binned read counts into several states which represent copy-numbers on each strand.

Usage

```
findCNVs.strandseq(binned.data, ID = NULL, R = 10, sig.lvl = 0.1,
  eps = 0.01, init = "standard", max.time = -1, max.iter = 1000,
  num.trials = 5, eps.try = max(10 * eps, 1), num.threads = 1,
  count.cutoff.quantile = 0.999, strand = "*",
  states = c("zero-inflation", paste0(0:10, "-somy")),
  most.frequent.state = "1-somy", method = "edivisive", algorithm = "EM",
  initial.params = NULL)
```

Arguments

binned.data	A GRanges-class 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.
R	method-edivisive: The maximum number of random permutations to use in each iteration of the permutation test (see e.divisive). Increase this value to increase accuracy on the cost of speed.
sig.lvl	method-edivisive: The level at which to sequentially test if a proposed change point is statistically significant (see e.divisive). Increase this value to find more breakpoints.
eps	method-HMM: Convergence threshold for the Baum-Welch algorithm.
init	method-HMM: 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	method-HMM: 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. Set max.time = -1 for no limit.
max.iter	method-HMM: The maximum number of iterations for the Baum-Welch algorithm. Set max.iter = -1 for no limit.
num.trials	method-HMM: 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	method-HMM: 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	method-HMM: Number of threads to use. Setting this to >1 may give increased performance.
count.cutoff.quantile	method-HMM: 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	Find copy-numbers only for the specified strand. One of c('+', '-', '*').
states	method-HMM: A subset or all of c("zero-inflation","0-somy","1-somy","2-somy","3-somy","4- 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	method-HMM: One of the states that were given in states. The specified state is assumed to be the most frequent one. This can help the fitting procedure to converge into the correct fit.

method	Any combination of c('HMM','dnacopy','edivisive'). Option method='HMM' uses a Hidden Markov Model as described in doi:10.1186/s13059-016-0971-7 to call copy numbers. Option 'dnacopy' uses segment from the **DNAcopy** package to call copy numbers similarly to the method proposed in doi:10.1038/nmeth.3578, which gives more robust but less sensitive results compared to the HMM. Option 'edivisive' (DEFAULT) works like option 'dnacopy' but uses the e.divisive function from the **ecp** package for segmentation.
algorithm	method-HMM: 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	method-HMM: A aneuHMM object or file containing such an object from which initial starting parameters will be extracted.

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 file into bin size 1Mp
binned <- binReads(bedfile, assembly='mm10', binsize=1e6,
                   chromosomes=c(1:19,'X','Y'), pairedEndReads=TRUE)
## Find copy-numbers
model <- findCNVs.strandseq(binned[[1]])
## Check the fit
plot(model, type='histogram')
plot(model, type='profile')
```

---

findHotspots *Find breakpoint hotspots*

Description

Find breakpoint hotspots with kernel density estimation (KDE).

Usage

```
findHotspots(models, bw, pval = 0.05, spacing.bp = 5000, filename = NULL)
```

Arguments

models	A list of GRanges-class or aneuHMM objects or a character vector with files that contain such objects.
bw	Bandwidth used for kernel density estimation (see density).
pval	P-value cutoff for hotspots.
spacing.bp	Spacing of datapoints for KDE in basepairs.
filename	Will write hotspot coordinates and densities to the specified file. Endings "_breakpoint-hotspots.bed.gz" and "_breakpoint-densities.wig.gz" will be appended to filename.

Details

findHotspots uses density to perform a KDE. A p-value is calculated by comparing the density profile of the genomic events with the density profile of a randomly subsampled set of genomic events. Due to this random sampling, the result can vary for each function call, most likely for hotspots whose p-value is close to the specified pval.

Value

A list of GRanges-class objects containing 1) coordinates of hotspots and 2) p-values within the hotspot.

---

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, stepsizes = NULL,
  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 getChromInfoFromUCSC for available assemblies. This option is ignored if bamfile is specified. Alternatively a data.frame generated by getChromInfoFromUCSC.
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.

stepsizes
: A vector of step sizes in base pairs, the same length as binsizes.

chromosomes
: A subset of chromosomes for which the bins are generated.

Value

A list() of GRanges-class objects with fixed-width bins. If stepsizes is specified, a list() of GRangesList objects with one entry per step.

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
```

---

getBreakpoints *Extract breakpoints*

Description

Extract breakpoints with confidence intervals from an aneuHMM or aneuBiHMM object.

Usage

```
getBreakpoints(model, fragments = NULL, confint = 0.99)
```

Arguments

model
: An aneuHMM or aneuBiHMM object or a file that contains such an object.

fragments
: A GRanges-class object with read fragments or a file that contains such an object.

confint
: Desired confidence interval for breakpoints. Set confint=NULL to disable confidence interval estimation.

Details

Confidence intervals for breakpoints are estimated by going outwards from the breakpoint read by read, and performing a test of getting the observed or a more extreme outcome, given that the reads within the confidence interval belong to the other side of the breakpoint.

Value

A GRanges-class with breakpoint coordinates and confidence interals if fragments was specified.

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 data into bin size 1Mp
readfragments <- binReads(bedfile, assembly='mm10', binsize=1e6,
                  chromosomes=c(1:19,'X','Y'), reads.return=TRUE)
binned <- binReads(bedfile, assembly='mm10', binsize=1e6,
                  chromosomes=c(1:19,'X','Y'))
## Fit the Hidden Markov Model
model <- findCNVs.strandseq(binned[[1]])
## Add confidence intervals
breakpoints <- getBreakpoints(model, readfragments)
```

---

getDistinctColors *Get distinct colors*

Description

Get a set of distinct colors selected from colors.

Usage

```
getDistinctColors(n, start.color = "blue4", exclude.colors = c("white",
  "black", "gray", "grey", "\\<yellow\\>", "yellow1", "lemonchiffon"),
  exclude.brightness.above = 1, exclude.rgb.above = 210)
```

Arguments

n	Number of colors to select. If n is a character vector, length(n) will be taken as the number of colors and the colors will be named by n.
start.color	Color to start the selection process from.
exclude.colors	Character vector with colors that should not be used.
exclude.brightness.above	Exclude colors where the 'brightness' value in HSV space is above. This is useful to obtain a matt palette.
exclude.rgb.above	Exclude colors where all RGB values are above. This is useful to exclude whitish colors.

Details

The function computes the euclidian distance between all colors and iteratively selects those that have the furthest closes distance to the set of already selected colors.

Value

A character vector with colors.

Author(s)

Aaron Taudt

Examples

```
cols <- AneuFinder:::getDistinctColors(5)
pie(rep(1,5), labels=cols, col=cols)
```

`getQC` *Obtain a data.frame with quality metrics*

Description

Obtain a data.frame with quality metrics from a list of `aneuHMM` objects or a list of files that contain such objects.

Usage

```
getQC(models)
```

Arguments

`models`	A list of `GRanges-class` or `aneuHMM` objects or a character vector with files that contain such objects.

Details

The employed quality measures are:

- total.read.count: Total read count.
- avg.binsize: Average binsize.
- avg.read.count: Average read count.
- spikiness: Bin-to-bin variability of read count.
- entropy: Shannon entropy of read counts.
- complexity: Library complexity approximated with a Michaelis-Menten curve.
- loglik: Loglikelihood of the Hidden Markov Model.
- num.segments: Number of copy number segments that have been found.
- bhattacharrya distance: Bhattacharyya distance between 1-somy and 2-somy distributions.
- sos: Sum-of-squares distance of read counts to the fitted distributions in their respective segments.

Value

A data.frame with columns

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)
df <- getQC(files)
```

---

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)
```

Arguments

model	An aneuBiHMM object.
resolution	An integer vector specifying the resolution at bin level at which to scan for SCE events.
min.segwidth	Segments below this width will be removed before scanning for SCE events.
fragments	A GRanges-class object with read fragments or a file that contains such an object. These reads will be used for fine mapping of the SCE events.

Value

A GRanges-class 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, assembly='hg19', binsize=1e6,
                   chromosomes=c(1:22,'X','Y'), pairedEndReads=TRUE)
## Fit the Hidden Markov Model
## Find copy-numbers
model <- findCNVs.strandseq(binned[[1]])
## Find sister chromatid exchanges
model$sce <- getSCEcoordinates(model)
print(model$sce)
plot(model)
```

`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 a character vector with 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.breakpoints = FALSE, hotspots = NULL,
  exclude.regions = NULL)
```

Arguments

`hmms`	A list of `aneuHMM` objects or a character vector with 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`.
`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.breakpoints`	Logical indicating whether breakpoints should be plotted.
`hotspots`	A `GRanges-class` object with coordinates of genomic hotspots (see `hotspotter`).
`exclude.regions`	A `GRanges-class` with regions that will be excluded from the computation of the clustering. This can be useful to exclude regions with artifacts.

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'))
```

---

heatmapGenomewideClusters

Plot heatmaps for quality control

Description

This function is a convenient wrapper to call heatmapGenomewide for all clusters after calling clusterByQuality and plot the heatmaps into one pdf for efficient comparison.

Usage

```
heatmapGenomewideClusters(cl = NULL, cutree = NULL, file = NULL, ...)
```

Arguments

cl	The return value of clusterByQuality.
cutree	The return value of cutree, where the names correspond to the filenames to be loaded.
file	A character specifying the output file.
...	Further parameters passed on to heatmapGenomewide.

Value

A cowplot object or NULL if a file was specified.

Examples

```
## Get a list of HMMs and cluster them
folder <- system.file("extdata", "primary-lung", "hmms", package="AneuFinderData")
files <- list.files(folder, full.names=TRUE)
cl <- clusterByQuality(files, G=5)
heatmapGenomewideClusters(cl=cl)

## Plot sub-clones of the largest cluster
largest.cluster <- which.max(sapply(cl$classification, length))
files <- cl$classification[[largest.cluster]]
clust <- clusterHMMs(files)
groups <- cutree(tree = clust$hclust, k = 5)
heatmapGenomewideClusters(cutree = groups, cluster = FALSE)
```

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

Description

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

Usage

```
HMM.findCNVs(binned.data, ID = NULL, eps = 0.01, 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,
  verbosity = 1)
```

Arguments

binned.data	A GRanges-class object with binned read counts. Alternatively a GRangesList object with offsetted 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	method-HMM: Convergence threshold for the Baum-Welch algorithm.
init	method-HMM: 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` method-HMM: 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. Set `max.time = -1` for no limit.

`max.iter` method-HMM: The maximum number of iterations for the Baum-Welch algorithm. Set `max.iter = -1` for no limit.

`num.trials` method-HMM: 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` method-HMM: 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` method-HMM: Number of threads to use. Setting this to >1 may give increased performance.

`count.cutoff.quantile` method-HMM: 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` Find copy-numbers only for the specified strand. One of `c('+', '-', '*')`.

`states` method-HMM: A subset or all of `c("zero-inflation","0-somy","1-somy","2-somy","3-somy","4-`
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` method-HMM: One of the states that were given in `states`. The specified state is assumed to be the most frequent one. This can help the fitting procedure to converge into the correct fit.

`algorithm` method-HMM: 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` method-HMM: A aneuHMM object or file containing such an object from which initial starting parameters will be extracted.

`verbosity` method-HMM: Integer specifying the verbosity of printed messages.

Value

An aneuHMM object.

`hotspotter` *Find hotspots of genomic events*

Description

Find hotspots of genomic events by using kernel density estimation.

Usage

```
hotspotter(breakpoints, bw, pval = 0.05, spacing.bp = 5000)
```

Arguments

breakpoints	A list with GRanges-class object containing the coordinates of the genomic events.
bw	Bandwidth used for kernel density estimation (see density).
pval	P-value cutoff for hotspots.
spacing.bp	Spacing of datapoints for KDE in basepairs.

Details

The hotspotter uses density to perform a KDE. A p-value is calculated by comparing the density profile of the genomic events with the density profile of a randomly subsampled set of genomic events (bootstrapping).

Value

A list of GRanges-class objects containing 1) coordinates of hotspots and 2) p-values within the hotspot.

Author(s)

Aaron Taudt

---

hotspotter.variable *Find hotspots of genomic events*

Description

Find hotspots of genomic events by using kernel density estimation.

Usage

```
hotspotter.variable(breakpoints, confint, pval = 0.05, spacing.bp = 5000)
```

Arguments

breakpoints	A list with GRanges-class object containing the coordinates of the genomic events and their confidence intervals.
confint	Confidence interval that was used for breakpoint estimation.
pval	P-value cutoff for hotspots.
spacing.bp	Spacing of datapoints for KDE in basepairs.

Details

The hotspotter uses a gaussian kernel with variable bandwidth to perform a KDE. The bandwidth depends on the confidence intervals of the breakpoints. A p-value is calculated by comparing the density profile of the genomic events with the density profile of a randomly subsampled set of genomic events (bootstrapping).

Value

A list of GRanges-class objects containing 1) coordinates of hotspots and 2) p-values within the hotspot.

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-class 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-class 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, regions = NULL,
  exclude.regions = NULL)
```

Arguments

hmms	A list with aneuHMM objects or a list of files that contain such objects.
normalChromosomeNumbers	A named integer vector or matrix with physiological copy numbers, where each element (vector) or column (matrix) corresponds to a chromosome. This is useful to specify male or female samples, e.g. c('X'=2) for female samples or c('X'=1,'Y'=1) for male samples. Specify a vector if all your hmms have the same physiological copy numbers. Specify a matrix if your hmms have different physiological copy numbers (e.g. a mix of male and female samples). If not specified otherwise, '2' will be assumed for all chromosomes.
regions	A GRanges-class object containing ranges for which the karyotype measures will be computed.
exclude.regions	A GRanges-class with regions that will be excluded from the computation of the karyotype measures. This can be useful to exclude regions with artifacts.

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. If region was specified, a third list entry $regions will contain the regions with karyotype measures.

Author(s)

Aaron Taudt

Examples

```
### Example 1 ###
## 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)
## Compare karyotype measures between the two cancers
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)

### Example 2 ###
## Construct a matrix with physiological copy numbers for a mix of 5 male and 5 female samples
normal.chrom.numbers <- matrix(2, nrow=10, ncol=24,
                      dimnames=list(sample=c(paste('male', 1:5), paste('female', 6:10)),
                                                 chromosome=c(1:22,'X','Y')))
normal.chrom.numbers[1:5,c('X','Y')] <- 1
normal.chrom.numbers[6:10,c('Y')] <- 0
print(normal.chrom.numbers)

### Example 3 ###
## Exclude artifact regions with high variance
consensus <- consensusSegments(c(lung.files, liver.files))
variance <- apply(consensus$copy.number, 1, var)
exclude.regions <- consensus[variance > quantile(variance, 0.999)]
## Compare karyotype measures between the two cancers
normal.chrom.numbers <- rep(2, 23)
names(normal.chrom.numbers) <- c(1:22,'X')
lung <- karyotypeMeasures(lung.files, normalChromosomeNumbers=normal.chrom.numbers,
                        exclude.regions = exclude.regions)
liver <- karyotypeMeasures(liver.files, normalChromosomeNumbers=normal.chrom.numbers,
                         exclude.regions = exclude.regions)
print(lung$genomewide)
print(liver$genomewide)
```

`loadFromFiles` *Load* **AneuFinder** *objects from file*

Description

Wrapper to load **AneuFinder** objects from file and check the class of the loaded objects.

Usage

```
loadFromFiles(files, check.class = c("GRanges", "GRangesList", "aneuHMM",
  "aneuBiHMM"))
```

Arguments

`files`	A list of `GRanges-class`, `GRangesList`, `aneuHMM` or `aneuBiHMM` objects or a character vector with files that contain such objects.
`check.class`	Any combination of `c('GRanges', 'GRangesList', 'aneuHMM', 'aneuBiHMM')`. If any of the loaded objects does not belong to the specified class, an error is thrown.

Value

A list of `GRanges-class`, `GRangesList`, `aneuHMM` or `aneuBiHMM` objects.

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 ten
hmms <- loadFromFiles(files[1:10])
lapply(hmms, plot, type='profile')
```

`mergeStrandseqFiles` *Merge Strand-seq libraries*

Description

Merge strand libraries to generate a high-coverage Strand-seq library.

Usage

```
mergeStrandseqFiles(files, assembly, chromosomes = NULL,
  pairedEndReads = FALSE, min.mapq = 10, remove.duplicate.reads = TRUE,
  max.fragment.width = 1000)
```

Arguments

`files`	A character vector with files with aligned reads.
`assembly`	Please see `getChromInfoFromUCSC` for available assemblies. Only necessary when importing BED files. BAM files are handled automatically. Alternatively a data.frame with columns 'chromosome' and 'length'.
`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).
`min.mapq`	Minimum mapping quality when importing from BAM files. Set `min.mapq=NA` 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 `GRanges-class` object with reads.

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-class 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.

`plot.GRangesList` *Plotting function for binned read counts (list)*

Description

Make plots for binned read counts (list) from `binned.data`.

Usage

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

Arguments

`x`	A `GRangesList` 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.

`plotHeterogeneity` *Heterogeneity vs. Aneuploidy*

Description

Make heterogeneity vs. aneuploidy plots using individual chromosomes as datapoints.

Usage

```
plotHeterogeneity(hmms, hmms.list = NULL, normalChromosomeNumbers = NULL,
  plot = TRUE, regions = NULL, exclude.regions = NULL)
```

Arguments

hmms	A list of aneuHMM objects or a character vector with files that contain such objects.
hmms.list	Alternative input for a faceted plot. A named list() of lists of aneuHMM objects. Alternatively a named list() of character vectors with files that contain aneuHMM objects. List names serve as facets for plotting. If specified, normalChromosomeNumbers is assumed to be a list() of vectors (or matrices) instead of a vector (or matrix).
normalChromosomeNumbers	A named integer vector or matrix with physiological copy numbers, where each element (vector) or column (matrix) corresponds to a chromosome. This is useful to specify male or female samples, e.g. c('X'=2) for female samples or c('X'=1,'Y'=1) for male samples. Specify a vector if all your hmms have the same physiological copy numbers. Specify a matrix if your hmms have different physiological copy numbers (e.g. a mix of male and female samples). If not specified otherwise, '2' will be assumed for all chromosomes. If you have specified hmms.list instead of hmms, normalChromosomeNumbers is assumed to be a list() of vectors (or matrices), with one vector (or matrix) for each element in hmms.list.
plot	A logical indicating whether to plot or to return the underlying data.frame.
regions	A GRanges-class object containing ranges for which the karyotype measures will be computed.
exclude.regions	A GRanges-class with regions that will be excluded from the computation of the karyotype measures. This can be useful to exclude regions with artifacts.

Value

A ggplot object or a data.frame if plot=FALSE.

Examples

```
### Example 1: A faceted plot of lung and liver cells ###
## 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)
## Make heterogeneity plots
plotHeterogeneity(hmms.list = list(lung=lung.files, liver=liver.files))

### Example 2: Plot a mixture sample of male and female cells ###
## Get results from a small-cell-lung-cancer
folder <- system.file("extdata", "primary-lung", "hmms", package="AneuFinderData")
files <- list.files(lung.folder, full.names=TRUE)
## Construct a matrix with physiological copy numbers for a mix of 48 male and 48 female samples
normal.chrom.numbers <- matrix(2, nrow=96, ncol=24,
                    dimnames=list(sample=c(paste('male', 1:48), paste('female', 49:96)),
                                  chromosome=c(1:22,'X','Y')))
```

```
normal.chrom.numbers[1:48,c('X','Y')] <- 1
normal.chrom.numbers[49:96,c('Y')] <- 0
head(normal.chrom.numbers)
## Make heterogeneity plots
plotHeterogeneity(hmms = files, normalChromosomeNumbers = normal.chrom.numbers)

### Example 3: A faceted plot of male lung and female liver cells ###
## 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)
## Specify the physiological copy numbers
chrom.numbers.lung <- c(rep(2, 22), 1, 1)
names(chrom.numbers.lung) <- c(1:22, 'X', 'Y')
print(chrom.numbers.lung)
## 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)
## Specify the physiological copy numbers
chrom.numbers.liver <- c(rep(2, 22), 2, 0)
names(chrom.numbers.liver) <- c(1:22, 'X', 'Y')
print(chrom.numbers.liver)
## Make heterogeneity plots
plotHeterogeneity(hmms.list = list(lung=lung.files, liver=liver.files),
                  normalChromosomeNumbers = list(chrom.numbers.lung, chrom.numbers.liver))

### Example 4 ###
## Exclude artifact regions with high variance
consensus <- consensusSegments(c(lung.files, liver.files))
variance <- apply(consensus$copy.number, 1, var)
exclude.regions <- consensus[variance > quantile(variance, 0.999)]
## Make heterogeneity plots
plotHeterogeneity(hmms.list = list(lung=lung.files, liver=liver.files),
                   exclude.regions=exclude.regions)
```

`plotHistogram` *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

```
plotHistogram(model)
```

Arguments

model	A aneuHMM object.

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.breakpoints = TRUE,
  file = NULL)
```

Arguments

model	A aneuHMM object or binned.data.
both.strands	If TRUE, strands will be plotted separately.
plot.breakpoints	Logical indicating whether breakpoints 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.breakpoints = FALSE,
  file = NULL, normalize.counts = NULL)
```

Arguments

model	A aneuHMM object or binned.data.
both.strands	If TRUE, strands will be plotted separately.
plot.breakpoints	Logical indicating whether breakpoints should be plotted.
file	A PDF file where the plot will be saved.
normalize.counts	An character giving the copy number state to which to normalize the counts, e.g. '1-somy', '2-somy' etc.

Value

A ggplot object or NULL if a file was specified.

---

plot_pca *Perform a PCA for copy number profiles*

Description

Perform a PCA for copy number profiles in aneuHMM objects.

Usage

```
plot_pca(hmms, PC1 = 1, PC2 = 2, colorBy = NULL, plot = TRUE,
  exclude.regions = NULL)
```

Arguments

hmms	A list of aneuHMM objects or a character vector with files that contain such objects.
PC1	Integer specifying the first of the principal components to plot.
PC2	Integer specifying the second of the principal components to plot.
colorBy	A character vector of the same length as hmms which is used to color the points in the plot.
plot	Set to FALSE if you want to return the data.frame that is used for plotting instead of the plot.
exclude.regions	A GRanges-class with regions that will be excluded from the computation of the PCA. This can be useful to exclude regions with artifacts.

Value

A ggplot object or a data.frame if plot=FALSE.

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 the PCA
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)))
plot_pca(c(lung.files, liver.files), colorBy=classes, PC1=2, PC2=4)
```

---

print.aneuBiHMM *Print aneuBiHMM object*

Description

Print aneuBiHMM object

Usage

```
## S3 method for class 'aneuBiHMM'
print(x, ...)
```

Arguments

x	An aneuBiHMM object.
...	Ignored.

Value

An invisible NULL.

---

print.aneuHMM *Print aneuHMM object*

Description

Print aneuHMM object

Usage

```
## S3 method for class 'aneuHMM'
print(x, ...)
```

Arguments

x	An aneuHMM object.
...	Ignored.

Value

An invisible NULL.

---

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)

qc.sos(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
- qc.sos: Sum-of-squares distance from the read counts to the fitted distributions

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

---

refineBreakpoints *Refine breakpoints*

Description

Refine breakpoints with confidence intervals from an initial estimate (from getBreakpoints).

Usage

```
refineBreakpoints(model, fragments, breakpoints = model$breakpoints,
  confint = 0.99)
```

Arguments

model	An aneuBiHMM object or a file that contains such an object.
fragments	A GRanges-class object with read fragments or a file that contains such an object.
breakpoints	A GRanges-class object with breakpoints and confidence intervals, as returned by function getBreakpoints.
confint	Desired confidence interval for breakpoints.

Details

Breakpoints are refined by shifting the breakpoint within its initial confidence interval read by read and maximizing the probability of observing the left-right read distribution.

Value

An aneuBiHMM with adjusted breakpoint coordinates and confidence interals, bins and segments.

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 data into bin size 1Mp
readfragments <- binReads(bedfile, assembly='mm10', binsize=1e6,
                  chromosomes=c(1:19,'X','Y'), reads.return=TRUE)
binned <- binReads(bedfile, assembly='mm10', binsize=1e6,
                  chromosomes=c(1:19,'X','Y'))
## Fit the Hidden Markov Model
model <- findCNVs.strandseq(binned[[1]])
## Add confidence intervals
breakpoints <- getBreakpoints(model, readfragments)
## Refine breakpoints
refined.model <- refineBreakpoints(model, readfragments, breakpoints)
```

---

simulateReads *Simulate reads from genome*

Description

Simulate single or paired end reads from any **BSgenome-class** 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-class** 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 a character vector with files that contain such objects.

profile
: A GRanges-class 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-class object. This is useful for making genomewide plots.

Usage

```
transCoord(gr)
```

Arguments

gr	A GRanges-class object.

Value

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

---

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, stepsizes = NULL, chromosomes = NULL)
```

Arguments

reads	A GRanges-class with reads. See bam2GRanges and bed2GRanges.
binsizes	A vector with binsizes. Resulting bins will be close to the specified binsizes.
stepsizes	A vector of step sizes in base pairs, the same length as 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-class objects with variable-width bins. If stepsizes is specified, a list() of GRangesList objects with one entry per step.

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[['binsize_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