Package 'kfltCNV'

January 17, 2019

Version 0.1.0

Date 2019-01-08

Title kfltCNV: A tool for calling CNVs (Copy number variations) by the Kalman Filter

Author Qi-Yuan Li,
Zhan-Ni Chen

Maintainer Zhan-Ni Chen <sanadamakomi@gmail.com>

Depends R (>= 3.4.0)

Imports methods,
utils,
stats,
grDevices,
graphics,
BiocGenerics,
S4Vectors,
IRanges,
GenomicRanges,
Biostrings,
GenomeInfoDb,
Rsamtools,
BiocParallel,
SummarizedExperiment,
GenomicAlignments,
nlme,
zoo,
xts,
FKF,
parallel,
ggplot2,
devtools

Suggests roxygen2,
knitr,
rmarkdown,
BiocStyle

Description CNVs (copy number variations) can be detected by read depth data from Next-generation Sequencing. Read depth is Gaussian so we can use the Kalman Filter to estimate the real level of copy number. The Kalman filter is a set of recursive linear estimation steps, which computes an estimation value at time 't+1' based on the estimation value at time 't' and the

observation value at time 't+1'. Its estimates are linear combinations of Gaussian data when the noise disturbances are Gaussian.

License Artistic-2.0

Encoding UTF-8

URL https://github.com/sanadamakomi/kfltCNV

BugReports https://github.com/sanadamakomi/kfltCNV/issues

VignetteBuilder BiocStyle, knitr, rmarkdown

RoxygenNote 6.1.1

R topics documented:

annovateBedFormat *Annovating BED format file with gene symbol.*

Description

Annovating BED format file with gene symbol. It will create a new file with a column named *gene* adding to input file.

Usage

```
annovateBedFormat(x, y, annote.database)
```

Arguments

x	A character string of file path.
y	A character string of a file to write to.
annote.database	A character string of *refGene.txt* file path. It can be download from UCSC database ftp. And *refGene.txt* from *Annovar* database is supported, too.

Author(s)

Zhan-Ni Chen

Examples

```
####### Annovating BED format file with gene symbol #######
annovateBedFormat(x = system.file("extdata", 'testSample.state', package = "kfltCNV"),
  y = 'test.anno.state',
 annote.database = system.file("extdata", 'hg19_refGene_chr10.txt', package = "kfltCNV"))
```

bedtoGRange *Read BED file and return a GenomicRanges object.*

Description

Read BED file and return a GenomicRanges object. Three required BED fields are chromosome name, start and end position.

Usage

```
bedtoGRange(file, genomeSeqinfo = NULL)
```

Arguments

file	A character string of the BED file path.
genomeSeqinfo	A GenomeInfoDb object which contain the sequence information. It can also be extracted from a BAM file by **Rsamtools** package `seqinfo(BamFile("bam_file_path"))`.

Value

A GenomicRanges object of input BED file.

Author(s)

Zhan-Ni Chen

Examples

```
####### Read BED file and return a GenomicRanges object. #######
require(GenomicRanges)
require(Rsamtools)
gr <- GRanges(Rle(c("1", "1", "2")),
  IRanges(start = c(1 ,10000, 100), end = c(1000, 100000, 10000)))
grangetoBed(gr, 'target.bed')
bedtoGRange('target.bed')

# genome seqinfo
bamSeqinfo <- seqinfo(
  BamFile(system.file("extdata", 'testSample.bam', package = "kfltCNV")))
bedtoGRange('target.bed', genomeSeqinfo = bamSeqinfo)
```

`calculateLog2ratio` *Compute the log2 ratio between sample depth and baseline.*

Description

Input vectors of sample depth and baseline to compute the log2 ratio.

Usage

```
calculateLog2ratio(x, baseline, badDepth)
```

Arguments

`x`	A numeric vector of sample depth.
`baseline`	A numeric vector contains baseline depth of region with the same order in sample depth vector.
`badDepth`	A numeric value; a baseline depth low than badDepth will be set to `NA`.

Value

A numeric vector of log2 ratio.

Author(s)

Zhan-Ni Chen

Examples

```
####### Compute log2ratio #######
covmatrix <- mergerCovFiles(c(
  system.file("extdata", 'testSample.cov', package = "kfltCNV"),
  system.file("extdata", 'controlSample.cov', package = "kfltCNV"),
  system.file("extdata", 'controlSample2.cov', package = "kfltCNV")))

normal.coveragematrix <- normalizeCovMatrix(covmatrix)
baseline.control <- createControlBaseline(normal.coveragematrix[, -4])
log2ratio <- calculateLog2ratio(normal.coveragematrix[, 4],
  baseline.control[, 4], badDepth = 10)
```

`callGainCNV` *Call CNVs.*

Description

Read a log2 ratio state file (<filename>.state) and call CNVs.

`callGainCNV()` will merge and output copy number gains.

`callLossCNV()` will merge and output copy number losses.

`callCNVGene()` will compute gene's copy number in the CNV segment. It need an input file with a column named *gene*. It can be created by a log2 ratio state file (<filename>.state) using the function `annovateBedFormat`.

Usage

```
callGainCNV(file, gapwidth)

callLossCNV(file, gapwidth)

callCNVGene(file, gapwidth)
```

Arguments

`file`	A character string of a log2 ratio state file (<filename>.state) path.
`gapwidth`	A non-negative integer indicates the maximum gap width between CNV region to ignore.

Details

The result is a data frame with nine columns and a *gene* column if `callCNVGene()` is used:

chr - chromosome name.

start - start position of CNV segment.

end - end position of CNV segment.

svtype - *DUP* or *DEL* representing copy number gains or losses in CNV segment.

log2ratio - log2 ratio of CNV segment.

probe - the number of bins covered by CNV segment.

depth - median depth of CNV segment in test sample.

baseline - median depth of CNV segment in baseline.

cn - copy number of CNV segment.

gene - gene symbol.

Value

If find no CNVs it will retrun NULL. Or else return a data frame with nine columns named *chr*, *start*, *end*, *svtype*, *log2ratio*, *probe*, *depth*, *baseline* and *cn*.

Author(s)

Zhan-Ni Chen

Examples

```
####### Call CNVs #######
callGainCNV(system.file("extdata", 'testSample.state', package = "kfltCNV"), gapwidth = 0)
```

`callGenderByBam` *Call gender from BAM file.*

Description

Call gender by read depth of chromosome X and Y. Compare the chi-squared values obtained to infer whether the male or female assumption fits read depth better.

Usage

```
callGenderByBam(bam, lower = 20)
```

Arguments

`bam`	A character string of BAM file path.
`lower`	A non-negative integer. Position of which coverage is lower than the integer will not be count.

Value

A character string, *Unknow*, *Female* or *Male*.

Author(s)

Zhan-Ni Chen

Examples

```
####### Call gender from BAM file #######
callGenderByBam(system.file("extdata", 'testSample.bam', package ="kfltCNV"))
```

callGenderByCov *Call gender from coverage file.*

Description

Call gender by read depth of chromosome X and Y. Compare the chi-squared values obtained to infer whether the male or female assumption fits read depth better.

Usage

```
callGenderByCov(x)
```

Arguments

x	A character string of coverage file (<fileName>.cov) path.

Value

A character string, *Unknow*, *Female* or *Male*.

Author(s)

Zhan-Ni Chen

Examples

```
####### Call gender from coverage file #######
callGenderByCov(system.file("extdata", 'testSample.cov', package = "kfltCNV"))
```

checkBam *Check BAM file.*

Description

It will stop if BAM file is illegal.

Usage

```
checkBam(x)
```

Arguments

x	A character string or vector of BAM File path.

Author(s)

Zhan-Ni Chen

Examples

```
####### Check BAM file #######
checkBam(system.file("extdata", 'testSample.bam', package = "kfltCNV"))
```

computeMinTotalBase *Compute minimum total bases of several coverage files.*

Description

Calculate minimum total bases of several coverage files (<filename>.cov). Four required fields in a coverage file are chromosome name, start, end, and depth of coverage.

Usage

```
computeMinTotalBase(x, method = "min", use.method = TRUE)
```

Arguments

x	A data frame or a matrix, the result of function 'mergerCovFiles'.
method	Default *min*, also can input *max*, *median*, or *mean*.
use.method	TRUE by default, if FALSE, return a numeric vector of total bases.

Value

A numeric value of total bases.

Author(s)

Zhan-Ni Chen

Examples

```
####### Compute minimum total bases of several coverage files #######
computeMinTotalBase(c(system.file("extdata", 'testSample.cov', package = "kfltCNV"),
 system.file("extdata", 'controlSample.cov', package = "kfltCNV")))#'

# output total base
computeMinTotalBase(c(system.file("extdata", 'testSample.cov', package = "kfltCNV"),
 system.file("extdata", 'controlSample.cov', package = "kfltCNV")), use.method = FALSE)
```

createBaseline *Create a baseline to call CNV.*

Description

Creating a baseline by *control*, *shuffle* or *movingAverage*.

Usage

```
createBaseline(x, path, mode = "control", total.base = NULL,
  step = 10, ifByChr = FALSE)
```

Arguments

x	A string value or vector of coverage file path.
path	A string value or vector of coverage file path.
mode	A character string, *control*, *shuffle* or *movingAverage*.
total.base	A non-negative numeric value used in mode *control* indicates a total base to scale to.
step	A non-negative integer used in mode *movingAverage*; it specifies the step by performing moving average.
ifByChr	A logical object used in mode *shuffle*; if TRUE, it will shuffle depth in each chromosome. If FALSE of the whole sample.

Details

The *control* mode (by default) will compute median depth of control samples in each region as the baseline. More than one coverage file (<filename>.cov) is need to pass and if total.base is null, it will scale to min total base.

The *shuffle* mode will shuffle the depth of each chromosome or the whole sample (when *isByChr* set to FALSE) to create a baseline. If several files input, the first one will be useing.

The *movingAverage* mode will compute moving average depth of each chromosome or the whole sample to create a baseline. If several files input, the first one will be useing.

Author(s)

Zhan-Ni Chen

Examples

```
####### Create a baseline to call CNV #######
# control
createBaseline(c(system.file("extdata", 'controlSample.cov', package = "kfltCNV"),
  system.file("extdata", 'controlSample2.cov', package = "kfltCNV")),
  path = 'baseline.control.cov')

# shuffle
createBaseline(system.file("extdata", 'testSample.cov', package = "kfltCNV"),
  mode = "shuffle", path = 'baseline.shuffle.cov')

# movingAverage
createBaseline(system.file("extdata", 'testSample.cov', package = "kfltCNV"),
  mode = "movingAverage", path = 'baseline.movingAverage.cov')
```

---

createControlBaseline *Create a baseline with control samples to call CNV.*

Description

Compute median depth of control samples in each region as the baseline for calling CNV.

Usage

```
createControlBaseline(x)
```

Arguments

x	A data frame or a matrix with the same format as the result of function mergerCovFiles, of which first three columns are chromosome, start and end, and following with several samples' depth. It returns a four-column data frame which has the same first three columns as input but the forth column is median depth of samples named *baseline*.

Value

A data frame with four columns named *chr*, *start*, *end*, *baseline*.

Author(s)

Zhan-Ni Chen

Examples

```
####### Create baseline to call CNV #######
covmatrix <- mergerCovFiles(c(
  system.file("extdata", 'testSample.cov', package = "kfltCNV"),
  system.file("extdata", 'controlSample.cov', package = "kfltCNV"),
  system.file("extdata", 'controlSample2.cov', package = "kfltCNV")))

normal.coveragematrix <- normalizeCovMatrix(covmatrix)
baseline.control <- createControlBaseline(normal.coveragematrix[, -4])
```

---

createMoveAveBaseline *Creat a baseline by moving average.*

Description

Compute moving average depth of each chromosome to create a baseline depth.

Usage

```
createMoveAveBaseline(x, step)
```

Arguments

x	a data frame or a matrix as the same format as the result of function mergerCovFiles, with first three columns of chromosome, start and end position, and forth column is depth to compute moving average. It returns a data frame of which first three columns are the same as input but the forth one is a moving average depth named *baseline*.
step	A non-negative integer; it specifies the step by performing moving average.

Value

A data frame with four columns.

Author(s)

Zhan-Ni Chen

Examples

```
####### Creat a baseline by moving average #######
tumor.coveragematrix <- read.table(
  system.file("extdata", 'testSample.cov', package = "kfltCNV"), header = FALSE)
colnames(tumor.coveragematrix) <- c('chr', 'start', 'end', 'testSample')
baseline.moveaverage <- createMoveAveBaseline(tumor.coveragematrix, step=10)
```

`createShuffleBaseline` *Creat a baseline by shuffling.*

Description

Shuffle the depth of each chromosome or the whole sample to create a baseline.

Usage

```
createShuffleBaseline(x, ifByChr = FALSE)
```

Arguments

`x`	A data frame or a matrix as the same format as the result of function `mergerCovFiles`, with first three columns of chromosome, start and end, and forth column is depth to shuffle. It returns a data frame of which first three columns are the same as input but the forth one is a shuffled depth named *baseline*.
`ifByChr`	Logical; if `TRUE`, it will shuffle depth of each chromosome. If `FALSE` of the whole sample.

Value

A data frame with four columns named *chr*, *start*, *end*, *baseline*.

Author(s)

Zhan-Ni Chen

Examples

```
####### Creat a baseline by shuffling #######
tumor.coveragematrix <- read.table(
  system.file("extdata", 'testSample.cov', package = "kfltCNV"), header = FALSE)
colnames(tumor.coveragematrix) <- c('chr', 'start', 'end', 'testSample')
baseline.shuffle <- createShuffleBaseline(tumor.coveragematrix)
```

depthOfRegion *Calculate depth of coverage in a given region.*

Description

Input a BAM file and a GenomicRanges object of given region, calculate the total bases overlapping this region. Sequencing depth of coverage of region equals total bases divide by width of region.

The input BAM file need a BAM index file (<filename>.bai). The index file can be generated by *samtools* or other BAM/SAM tools after sorting. Function will stop if the BAM file lose an index file or its index file is older than its own.

`depthOfRegionMultiCore()` can perform in multithreading with number of threads you input. Multithreading uses `parLapply()` function in **parallel** package and it can run on Linux but not Windows system. And the *batch* option limits the number of GRanges in a batch to run in a thread.

Usage

```
depthOfRegion(region, bamPath, mapq.filter = 30)

depthOfRegionMultiCore(region, bamPath, thread, batch, mapq.filter = 30,
  tmpDir = NULL)
```

Arguments

`region`	a GenomicRanges object.
`bamPath`	a character string of the BAM file path.
`mapq.filter`	a non-negative integer specifying the minimum mapping quality to include. BAM records with mapping qualities less than mapqFilter are discarded.
`thread`	an integer providing the number of threads just for `depthOfRegionMultiCore()`.
`batch`	an integer giving how many GRanges are performed in a batch just for `depthOfRegionMultiCore()`.
`tmpDir`	a character string of the directory path to write to.

Value

A GenomicRanges object with a column named depth representing the sequencing depth of this region.

Author(s)

Zhan-Ni Chen

Examples

```
####### Calculate depth of coverage in a given region #######
require(GenomicRanges)
depthOfRegion(
region = GRanges(Rle("10"), IRanges(start = 27444268, end = 27444388)),
bamPath = system.file("extdata", 'testSample.bam', package = "kfltCNV"))
```

grangetoBed *Write a GenomicRanges object into a BED file.*

Description

Write a GenomicRanges object into a BED file with three required BED fields, i.e. chromosome name, start and end position.

Usage

```
grangetoBed(gr, path)
```

Arguments

gr	A GenomicRanges object.
path	A character string of the BED file path to write to.

Author(s)

Zhan-Ni Chen

Examples

```
####### Write a GenomicRanges object into a BED file #######
require(GenomicRanges)
gr <- GRanges(Rle(c("1", "1", "2")),
  IRanges(start = c(1 ,10000, 100), end = c(1000, 100000, 10000)))
grangetoBed(gr, 'target.bed')
```

isSameBedFile *Test two BED files for equal.*

Description

Read two BED files and change them to sorted GenomicRanges objects, then use `identical()` to test for exactly equal. It returns `TRUE` in this case, `FALSE` in every other case. Three required BED fields are chromosome name, start and end position.

Usage

```
isSameBedFile(x, y)
```

Arguments

x, y	Character strings, a BED file or a file with first three columns the same format as a BED file, such as a coverage file (<filename>.cov).

Value

It returns `TRUE` if two BED file contain the same regions, `FALSE` in every other case.

Author(s)

Zhan-Ni Chen

Examples

```
####### Test two BED files for equal #######
isSameBedFile(x = system.file("extdata", 'chr10_exome.bed', package = "kfltCNV"),
y = system.file("extdata", 'testSample.cov', package = "kfltCNV"))
```

---

kflt *Kalman filter.*

Description

This function allows for Kalman filtering a numeric vector use the `fkf()` function in the package **FKF**. See the details in the funtion `fkf`.

Usage

```
kflt(yt, ct, model = list(p = 0, d = 1, q = 5, bin.size = 1e+05, sn =
  2.5, method = "nlminb"), nafun = na.locf, smooth = 0)
```

Arguments

`yt`	A matrix containing the observations. `NA` values are allowed.
`ct`	A matrix giving the intercept of the measurement equation.
`model`	A list object contains elements named *p*, *d*, *q*, *bin.size*, *sn* and *method*. See details in the function `kflt`. By default it is for log2 ratio data.
`nafun`	A function's name to fill `NA` value. Default is `na.locf`.
`smooth`	A non-negative integer indicating the step width to smooth the state after Kalman filtering.

Details

The Kalman filter is a set of recursive linear estimation steps, which computes an estimation value at time *'t+1'* based on the estimation value at time *'t'* and the observation value at time *'t+1'*. Its estimates are linear combinations of Gaussian data when the noise disturbances are Gaussian.

Read depth data from Next-Generation Sequencing are proved to be a combinations of Gaussian data. Perform log-ratio transformation on read depth to reduce system error from sequencing process, and smooth by moving-average. So the log2 ratio data is Gaussian and can be simulated by the Kalman filter.

Based on the Box-Jenkins Approach, the Log2 ratio data fits a model of *ARMA(0,5)-process* with zero (p) autoregressive term and five (q) moving-average terms.

Ande parameters in detail are:

yt - a matrix containing the observation and `NA` values are allowed. It is a *1 * n* matrix with log2 ratio data of n region.

ct - a matrix giving the intercept of the measurement equation. It is a *1 * 1* matrix and value is 0 indicating that log2 ratio is symmetrical around 0.

model - a list object containing six elements:

p - a non-negative integer providing the number of autoregressive terms.

q - a non-negative integer providing the number of moving-average terms.

d - a non-negative integer. If d is over 0, it will difference the data to reduce the changes in the level.

bin.size - a non-negative integer indicates the width of each window.

sn - a numberic value providing an average signal-to-noise ratio. If input data is unstable it can be turn down.

method - the method to optimize the model. By default it is *nlminb* which use the function `nlminb`. Other methods are *Nelder-Mead*, *BFGS*, *CG*, *L-BFGS-B*, *SANN* and *Brent* which use the function `optim`. See details in description of the function `optim`.

nafun - the function name in the package **zoo** to fill a `NA` data point. By default is `na.locf`.

smooth - an non-negative integer indicates the step when do moving-average. If smooth is 0, it return the original state of Kalman filtering.

Author(s)

Qi-Yuan Li

Examples

```
####### Kalman filter #######
kflt(
yt = matrix(data = log2(rnorm(1000, mean = 2, sd = 0.1)/2), nrow = 1),
ct = matrix(data = 1, nrow = 1))
```

`kfltBatch` *Batch pipeline of calling CNV.*

Description

A pipeline for using BAM files or baseline coverage files to call CNV by the Kalman Filter.

Usage

```
kfltBatch(testBamFile, controlBamFile = NULL, baselineFile = NULL,
  mode = "control", bedFile = NULL, binSize = NULL, outDir = NULL,
  annote.database = NULL, thread = 1, gapwidth = 0, threshold = 0,
  min.probes = 1, is.plot = TRUE, is.run.by.gender = FALSE,
  shuffle.ifByChr = FALSE, moveAverage.step = 10)
```

Arguments

`testBamFile`	A character string of test BAM file path.
`controlBamFile`	A character string or vector of control BAM files path.
`baselineFile`	A character string of baseline coverage file path.
`mode`	A character string, *control*, *shuffle* or *movingAverage*.
`bedFile`	A character string of BED file path.

`binSize` A non-negative integer giving the width of each bin or window, such as **1E2**, **1e2** or **100**.

`outDir` A character string of directory path to write to.

`annote.database` A character string of *refGene.txt* file path. It can be download from UCSC database ftp. And *refGene.txt* from *Annovar* database is supported, too.

`thread` A non-negative integer providing the number of threads.

`gapwidth` A non-negative integer indicates the maximum gap width between CNV region to ignore.

`threshold` A non-negative numeric value. A CNV of which log2 ratio is out of range from inverse *threshold* to *threshold* will not export.

`min.probes` A non-negative integer. A CNV of which probe is under *min.probes* will not export.

`is.plot` Logical. Whether to plot results.

`is.run.by.gender` Logial. Whether to call CNV based on sample's gender.

`shuffle.ifByChr` A logical object used in mode *shuffle*; if `TRUE`, it will shuffle depth of each chromosome. If `FALSE` of the whole sample.

`moveAverage.step` A non-negative integer used in mode *movingAverage*; it specifies the step by performing moving average.

Author(s)

Zhan-Ni Chen

Examples

```
####### Batch pipeline of calling CNV #######
testBamFile <- system.file("extdata", 'testSample.bam', package ="kfltCNV")
controlBamFile <- c(system.file("extdata", 'controlSample.bam', package ="kfltCNV"),
  system.file("extdata", 'controlSample2.bam', package ="kfltCNV"))
bedFile <- system.file("extdata", 'chr10_exome.bed', package ="kfltCNV")
binSize <- 1E2
kfltBatch(testBamFile, controlBamFile = controlBamFile, bedFile = bedFile, binSize = 1E2)
```

---

linearInterpolationCov

Align a coverage file.

Description

Align a coverage file (<filename>.cov) by a BED-like file and fit `NA` data point by linear interpolation.

Usage

```
linearInterpolationCov(covFile, axisFile)
```

Arguments

covFile	A character string of a coverage file path.
axisFile	A character string of a BED or BED-like file with first three columns of chromosome name, start and end position.

Details

Four required fields in a coverage file are chromosome name, start and end position, and depth of coverage. Three required BED-like fields are chromosome name, start and end position.

Value

A GenomicRanges object of which region is consistent with *axisFile* and has an updated depth column.

Author(s)

Zhan-Ni Chen

Examples

```
####### Align a coverage file #######
require(GenomicRanges)
gr <- readCovFile(system.file("extdata", 'testSample.cov', package = "kfltCNV"))
writeCovFile(shift(gr, shift = 10L), 'shift.cov')

linearInterpolationCov(covFile = 'shift.cov',
axisFile = system.file("extdata", 'testSample.cov', package = "kfltCNV"))
```

---

mergerCovFiles *Merge coverage files.*

Description

Align and merge coverage files (<filename>.cov) with chromosome, start and end position. Four required fields in a coverage file are chromosome name, start and end position, and depth of coverage.

Usage

```
mergerCovFiles(x)
```

Arguments

x	A character vector contains several coverage files path.

Value

A data frame, of which columns are chromosome, start position, end position and depths in input coverage files.

Author(s)

Zhan-Ni Chen

Examples

```
####### Merge coverage files #######
mergerCovFiles(c(system.file("extdata", 'testSample.cov', package = "kfltCNV"),
  system.file("extdata", 'controlSample.cov', package = "kfltCNV")))
```

---

normalizeCovMatrix *Normalize a coverage matrix.*

Description

Normalize every sample's coverage to the same total bases, i.e. the minimum total coverage of input samples. Input the result of function `mergerCovFiles`, which is a data frame with first three columns of chromosome, start, end, and following by columns of several samples' depths, and each row means different regions.

Usage

```
normalizeCovMatrix(x, total.base = NULL)
```

Arguments

`x`	A data frame or a matrix, the result of function `mergerCovFiles()`.
`total.base`	A non-negative integer, indicates a total base to scale to.

Value

A data frame with the same format as input.

Author(s)

Zhan-Ni Chen

Examples

```
####### Normalize a coverage matrix #######
covmatrix <- mergerCovFiles(c(system.file("extdata", 'testSample.cov', package = "kfltCNV"),
  system.file("extdata", 'controlSample.cov', package = "kfltCNV")))
normalizeCovMatrix(covmatrix)
```

outputTable *Export CNVs result.*

Description

Write CNVs result to tabular or VCF format file

Usage

```
outputTable(x, filepath, info.vec, threshold = 0.5, min.probes = 30)

outputVcf(x, filepath, id, info.vec, threshold = 0.5, min.probes = 30,
  addinfo = NULL, addformat = NULL)
```

Arguments

x	A data frame which is result of calling CNVs.
filepath	A character string of file path to write to.
info.vec	A character vector of column names in x to export.
threshold	A non-negative numeric value. A CNV of which log2 ratio is out of range from inverse *threshold* to *threshold* will not export.
min.probes	A non-negative integer. A CNV of which probe is under min.probes will not export.
id	A character string of the sample id in VCF file.
addinfo	A character string or vecter of column names in x to add to VCF file header and *INFO* column. Default header can be created by makeVcfHeader().
addformat	A character string or vecter of column names in x to add to VCF file header, *FORMAT* and sample column. Default header can be created by running makeVcfHeader.

Author(s)

Zhan-Ni Chen

Examples

```
loss <- callLossCNV(system.file("extdata", 'testSample.state', package = "kfltCNV"), 0)
gene <- callCNVGene(system.file("extdata", 'testSample.anno.state', package = "kfltCNV"), 0)

# tabular
outputTable(loss, filepath = 'cnv.result.txt',
  info.vec =  c("svtype", "svlen", "end", "log2ratio", "probe", "depth", "baseline"))

# vcf file
outputVcf(loss, filepath = 'cnv.result.vcf', id = 'testSample',
  info.vec =  c("svtype", "svlen", "end", "log2ratio", "probe", "depth", "baseline"))

# vcf file adding gene
outputVcf(gene, filepath = 'cnv.result.vcf', id = 'testSample',
  info.vec =  c("svtype", "end", 'gene', "log2ratio", "probe", "depth", "baseline"),
  addinfo = '##INFO=<ID=GENE,Number=1,Type=String,Description="Gene symbol">')
```

`performCallCNV` *Perform call CNVs.*

Description

Perform call CNVs from log2 ratio state file (<filename>.state). If *annote.database* isn't `NULL`, it will create the gene CNV result.

Usage

```
performCallCNV(file, outPrefix = NULL, id = NULL, gapwidth = 500,
  annote.database = NULL, out.type = "all", threshold = 0,
  min.probes = 1)
```

Arguments

`file`	A character string of a log2 ratio state file (<filename>.state) path.
`outPrefix`	A character string, indicates output files' prefix. By default it is the prefix of input file.
`id`	A character string of sample id in VCF file. By default it is the prefix of input file.
`gapwidth`	A non-negative integer indicates the maximum gap width between CNV region to ignore.
`annote.database`	A character string of *refGene.txt* file path. It can be download from UCSC database ftp. And *refGene.txt* from *Annovar* database is supported, too.
`out.type`	A character string, *vcf*, *table* or *all*. By default it will export both VCF file and tabular file.
`threshold`	A non-negative numeric value. A CNV of which log2 ratio is out of range from inverse *threshold* to *threshold* will not export.
`min.probes`	A non-negative integer. A CNV of which probe is under *min.probes* will not export.

Details

The results are tabular or VCFv4.2 format file.

The tabular file has nine columns while a gene CNV result will also have a column named *gene*.

chr - chromosome name.

start - start position of CNV segment.

end - end position of CNV segment.

svtype - *DUP* or *DEL* representing copy number gains or losses in CNV segment.

log2ratio - log2 ratio of CNV segment.

probe - the number of bins covered by CNV segment.

depth - median depth of CNV segment in test sample.

baseline - median depth of CNV segment in baseline.

cn - copy number of CNV segment.

gene - gene symbol.

Author(s)

Zhan-Ni Chen

Examples

```
####### Perform call CNVs #######
performCallCNV(
    file =  system.file("extdata", 'testSample.state', package = "kfltCNV"),
   annote.database = system.file("extdata", 'hg19_refGene_chr10.txt', package = "kfltCNV"))
```

---

performCreateCovFile *Perform calculating depth of coverage.*

Description

Input a BAM file or files to calculate the total bases overlapping the region. Sequencing depth of coverage in this region equals to total bases divide by width of region.

Usage

```
performCreateCovFile(bamFiles, bedFile = NULL, outDir = NULL,
  width = NULL, thread = 1, batch = 1000, mapq.filter = 30)
```

Arguments

bamFiles	A character string or vector of the BAM files path.
bedFile	A BED file path.
outDir	A character string of directory to output coverage files (<filename>.cov). Default is the current folder.
width	An integer giving the width of each window, such as **1E2**, **1e2** or **100**.
thread	An integer providing the number of thread.
batch	An integer giving how many GRanges are performed in a batch.
mapq.filter	A non-negative integer specifying the minimum mapping quality to include. BAM reads with mapping qualities less than mapqFilter are discarded.

Details

The input BAM file need a BAM index file (<filename>.bai). The index file can be generated by *samtools* or other BAM/SAM tools after sorting. Function will stop if the BAM file lose an index file or its index file is older than its own.

If *bedFile* is not NULL, it will be splited unless the *width* option is also NULL. If a BED file is not provided, windows will be created from the whole genome, by default 100,000 bp each window. Coverage of depth will be calculated by each window. Three required BED fields are chromosome name, start and end position.

It can perform in multithreading with number of threads you input. If the *thread* is over 1, it will perform in multithreading, default by single thread. Multithreading uses parLapply() function in **parallel** package and it can run on Linux but not Windows system. And the *batch* option limits the number of GRanges in a batch to run in a thread.

The result are coverage files (<filename>.cov) which have four columns indicating chromosome name, chromosome start, chromosome end and depth of coverage, without a header line.

Author(s)

Zhan-Ni Chen

Examples

```
####### Perform calculating depth of coverage #######
performCreateCovFile(
  bamFiles = system.file("extdata", 'testSample.bam', package = "kfltCNV"),
  bedFile = system.file("extdata", 'chr10_exome.bed', package = "kfltCNV"))
```

---

performFitCovFile *Prepare a log2 ratio file.*

Description

Prepare a log2 ratio file (<filename>.fit) to do Kalman filtering.

Usage

```
performFitCovFile(testFile, path, baselineFile = NULL,
  controlFile = NULL, badDepth = 10)
```

Arguments

testFile	A character string of a coverage file path to call CNV.
path	A character string of the log2 ratio file to output.
baselineFile	A character string of a baseline coverage file.
controlFile	A character string or vector of control samples' coverage files.
badDepth	A non-negative numeric value. The region of which depth in baseline low than badDepth will be set to NA.

Details

A log2 ratio file has six columns respectively named *chr*, *start*, *end*, *log2ratio*, *depth* and *baseline*.

And *log2ratio* is the log2 ratio of test sample's depth to the baseline. The columns named *depth* and *baseline* are depth of coverage respectively in test sample and the baseline.

If *baselineFile* isn't NULL, it will use baseline coverage file to create the log2 ratio file. When you input control samples' coverage files, it will use control samples to create a baseline at first.

Author(s)

Zhan-Ni Chen

Examples

```
####### Prepare a log2 ratio file #######
testFile <- system.file("extdata", 'testSample.cov', package = "kfltCNV")
controlFile <- c(system.file("extdata", 'controlSample.cov', package = "kfltCNV"),
  system.file("extdata", 'controlSample2.cov', package = "kfltCNV"))

performFitCovFile(testFile, path = 'testSample.fit', controlFile = controlFile)
```

performRunKflt *Perform Kalman filtering to estimate the log2 ratio level.*

Description

Use the Kalman filter to estimate a log2 ratio level and mark regions that significantly different from average level as copy number variantions.

Usage

```
performRunKflt(file, binSize, outPrefix = NULL, model = list(p = 0, d =
  1, q = 5, bin.size = NULL, sn = 2.5, method = "nlminb"), s = 7,
  alpha = 0.05, thread = 1, tmpDir = NULL)

runKflt(x, model = list(p = 0, d = 1, q = 5, bin.size = 1e+05, sn = 2.5,
  method = "nlminb"), s, alpha)

runKfltMultiCore(x, model = list(p = 0, d = 1, q = 5, bin.size = 1e+05,
  sn = 2.5, method = "nlminb"), s, alpha, thread, tmpDir = NULL)
```

Arguments

file	A string value of a log2 ratio file (<filename>.fit) path to do Kalman filtering.
binSize	A non-negative integer giving the width of each bin or window, such as **1E2**, **1e2** or **100**.
outPrefix	A character string, indicates output files' prefix.
model	A list object contains elements named *p*, *d*, *q*, *bin.size*, *sn*, and *method*. See details in the function kflt. By default it is for log2 ratio data.
s	A non-negative integer indicating the step width to smooth the state after Kalman filtering.
alpha	A numeric value providing the significant level.
thread	A non-negative integer providing the number of threads.
tmpDir	A character string of the directory path to write to.
x	A data frame of a log2 ratio file (<filename>.fit).

Details

The input file is a log2 ratio file (<filename>.fit) which has six columns respectively named *chr*, *start*, *end*, *log2ratio*, *depth* and *baseline*. And *log2ratio* is the log2 ratio of test sample's depth to baseline. The columns named *depth* and *baseline* are depth of coverage respectively in test sample and the baseline.

Results are two files named by prefix.

One is a estimated log2 ratio state file (<filename>.state). It has ten columns and the first six columns are from input log2 ratio file (<filename>.fit). Then another four columns are estimated log2 ratio state by the Kalman filter:

state - the log2 ratio state after kalman filtering.

statUp - the upper limmit of state confidence interval.

statDown - the lower limmit of state confidence interval.

report - the label representing the copy number variation level of a region. It has three levels i.e. gain, loss and average.

The other file is a parameter file (<filename>.parameter). It stores a best set of parameters found by `optim()`. `performRunKflt()` use the `fkf()` function in the package **FKF** to perform Kalman filtering and `optim()` to estimate parameters. By default it will simulate a **ARMA(0,5)-process**, so a parameter file has 6 columns named *par1* to *par6* representing six parameters in ARMA process, while the first column is chromosome name because it performs Kalman filtering by chromosome. See the details in the function `kflt`.

It can perform in multithreading with number of threads you input. If the *thread* is over 1, it will perform in multithreading, default by single thread. `runKflt()` and `runKfltMultiCore()` can run from a data frame and return a list of result.

Author(s)

Zhan-Ni Chen

Examples

```
####### Perform Kalman filtering to estimate the log2 ratio level #######
fitFile <- system.file("extdata", 'testSample.fit', package = "kfltCNV")
performRunKflt(fitFile, 'testSample', binSize = 1E2)

fit.dat <- read.delim(fitFile)
runKflt(fit.dat, s = 7, alpha = 0.05)
```

`plotCoverageUniformity` *Plot the unifomity of coverage among samples.*

Description

Compute correlation between columns of a numeric matrix or data frame and plot the distribution of correlations.

Usage

```
plotCoverageUniformity(x)
```

Arguments

x	A numeric matrix or a data frame.

Value

A boxplot R object.

Author(s)

Zhan-Ni Chen

Examples

```
####### Plot the coverage unifomity of samples input #######
covmatrix <- mergerCovFiles(c(system.file("extdata", 'testSample.cov', package = "kfltCNV"),
  system.file("extdata", 'controlSample.cov', package = "kfltCNV"),
  system.file("extdata", 'controlSample2.cov', package = "kfltCNV")))
plotCoverageUniformity(covmatrix[,4:6])
```

`plotKfltResult` *Plot kflt results.*

Description

Plot the log2 ratio state after Kalman filtering.

Usage

```
plotKfltResult(file)
```

Arguments

`file`	A character string of Kalman filter result file (<fileName>.state).

Author(s)

Zhan-Ni Chen

Examples

```
####### Plot kflt results #######
plotKfltResult(system.file("extdata", 'testSample.state', package = "kfltCNV"))
```

`readCovFile` *Read a coverage file.*

Description

Read a coverage file (<filename>.cov) and change it to a sorted GenomicRanges object. Four required a coverage file fields are chromosome name, start, end and coverage of depth.

Usage

```
readCovFile(file)
```

Arguments

`file`	A character string of coverage file path.

Value

A GenomicRanges object of genome region with a column named depth.

Author(s)

Zhan-Ni Chen

Examples

```
####### Read a coverage file #######
readCovFile(system.file("extdata", 'testSample.cov', package = "kfltCNV"))
```

---

scaleCovFile *Scale a coverage file to a provided total base.*

Description

Scale a coverage file (<filename>.cov) by a provided total base. Four required fields in a coverage file are chromosome name, start and end position, and depth of coverage.

Usage

```
scaleCovFile(x, path, total.base)
```

Arguments

x	A character string of a coverage file path.
path	A character string of a file to write to.
total.base	A non-negative integer, indicates a total base to scale to.

Author(s)

Zhan-Ni Chen

Examples

```
####### Scale a coverage file to a provided total base #######
scaleCovFile(system.file("extdata", 'testSample.cov', package = "kfltCNV"),
  path = 'testSample.nor.cov', total.base = 1E6)
```

---

splitBed *Spliting a BED file into windows.*

Description

Read a BED file to a GenomicRanges object and split it into windows with provided width. Three required BED fields are chromosome name, start and end position.

Usage

```
splitBed(file, width, genomeSeqinfo = NULL)
```

Arguments

file	A character string of a BED file path.
width	An integer giving the width of each window, such as **1E2**, **1e2** or **100**.
genomeSeqinfo	A GenomeInfoDb object which contains sequence information. It can also be extracted from a BAM file by **Rsamtools** package `seqinfo(BamFile("bam_file_path"))`.

Value

A GenomicRanges object of splited regions from the BED file input.

Author(s)

Zhan-Ni Chen

Examples

```
####### Spliting a BED file into windows #######
require(GenomicRanges)
require(Rsamtools)
gr <- GRanges(Rle(c("1", "1", "2")),
  IRanges(start = c(1 ,10000, 100), end = c(1000, 100000, 10000)))
grangetoBed(gr, 'target.bed')
splitBed('target.bed', 1E2)

# genome seqinfo
bamSeqinfo <- seqinfo(BamFile(system.file("extdata", 'testSample.bam', package = "kfltCNV")))
splitBed('target.bed', 1E2, genomeSeqinfo = bamSeqinfo)
```

---

splitGenome *Spliting genome into windows.*

Description

Split genome into windows with provided width.

Usage

```
splitGenome(genomeSeqinfo, width)
```

Arguments

genomeSeqinfo	A GenomeInfoDb object which contain the sequence information. It can also be extracted from a BAM file by **Rsamtools** package `seqinfo(BamFile("bam_file_path"))`.
width	An integer giving the width of each window, such as **1E2**, **1e2** or **100**.

Value

A GenomicRanges object of splited regions from genome.

Author(s)

Zhan-Ni Chen

Examples

```
####### Spliting genome into windows #######
require(Rsamtools)
bamSeqinfo <- seqinfo(BamFile(system.file("extdata", 'testSample.bam', package = "kfltCNV")))
splitGenome <- splitGenome(bamSeqinfo, width = 1E7)
```

`writeCovFile` *Write a GenomicRanges object into a coverage file.*

Description

Write a GenomicRanges object with a column named depth into a coverage file (<filename>.cov). The result file does not have a header line, and its four columns respectively indicate chromosome name, start position, end position, and depth.

Usage

```
writeCovFile(gr, path)
```

Arguments

`gr`	A GenomicRanges object with a column named depth.
`path`	A character string of a coverage file path to write to.

Author(s)

Zhan-Ni Chen

Examples

```
####### Write a GenomicRanges object into a coverage file #######
gr <- readCovFile(system.file("extdata", 'testSample.cov', package = "kfltCNV"))
writeCovFile(gr, 'sample.cov')
```

Index