# Package ‘AllelicImbalance’

March 22, 2017

**Type** Package  
**Title** Investigates Allele Specific Expression  
**Version** 1.12.0  
**Date** 2016-05-12  
**Encoding** UTF-8  
**Author** Jesper R Gadin, Lasse Folkersen  
**Maintainer** Jesper R Gadin <j.r.gadin@gmail.com>  
**Description** Provides a framework for allelic specific expression investigation using RNA-seq data.  
**License** GPL-3  
**URL** [https://github.com/pappewaio/AllelicImbalance](https://github.com/pappewaio/AllelicImbalance)  
**BugReports** [https://github.com/pappewaio/AllelicImbalance/issues](https://github.com/pappewaio/AllelicImbalance/issues)  
**Suggests** testthat, org.Hs.eg.db, TxDb.Hsapiens.UCSC.hg19.knownGene, SNPlocs.Hsapiens.dbSNP144.GRCh37, BiocStyle, knitr, rmarkdown  
**Depends** R (>= 3.2.0), grid, GenomicRanges, SummarizedExperiment (>= 0.2.0), GenomicAlignments  
**Imports** methods, BiocGenerics, AnnotationDbi, BSgenome, VariantAnnotation, Biostrings, S4Vectors (>= 0.9.25), IRanges, Rsamtools, GenomicFeatures, Gviz, lattice, latticeExtra, gridExtra, seqinr, GenomeInfoDb  
**LazyData** TRUE  
**biocViews** Genetics, Infrastructure, Sequencing  
**VignetteBuilder** knitr  
**Collate** 'AllelicImbalance-package.R' 'initialize-methods.R'  
'ASEset-class.R' 'DetectedAI-class.R' 'GlobalAnalysis-class.R'  
'barplot-methods.R' 'locationplot-methods.R'  
'GvizTrack-methods.R' 'LinkVariantAlmlof-class.R'  
'RegionSummary-class.R' 'RiskVariant-class.R'  
'auxillary-functions-annotation.R'  
'auxillary-functions-visuals.R'  
'auxillary-methods-annotation.R'  
'auxillary-methods-summaries.R' 'auxillary-methods.R'  
'chisq.test-methods.R' 'binom.test-methods.R'  
'boxplot-methods.R' 'deprecations.R' 'detect-methods.R'
RoxygenNote 5.0.1
NeedsCompilation no

R topics documented:

AllelicImbalance-package  .................................................. 3
annotation-wrappers  ................................................................ 4
annotationBarplot  ................................................................. 6
ASEset-barplot  ................................................................. 7
ASEset-class  ................................................................. 9
ASEset-filters  ............................................................ 14
ASEset-gbarplot  ........................................................... 15
ASEset-glocationplot  ................................................... 16
ASEset-gviztrack  ........................................................... 17
ASEset-locationplot  ......................................................... 18
ASEset-scanForHeterozygotes ........................................ 20
ASEset.old  ................................................................. 21
ASEset.sim  ................................................................. 22
ASEsetFromBam  ............................................................ 22
barplot-lattice-support ................................................. 23
binom.test  ................................................................. 24
chisq.test  ................................................................. 25
countAllelesFromBam .................................................. 26
coverageMatrixListFromGAL ............................................ 27
defaultMapBias ............................................................ 28
defaultPhase ................................................................. 29
detectAI  ................................................................. 29
DetectedAI-class .......................................................... 31
DetectedAI-plot ............................................................ 32
DetectedAI-summary ...................................................... 34
fractionPlotDf ............................................................... 35
gba ................................................................. 36
genomatrix ................................................................. 37
genotype2phase ............................................................ 37
getAlleleCounts ............................................................ 38
getAlleleQuality ............................................................ 39
getAreaFromGeneNames ................................................ 41
defaultMapBiasExpMean ............................................... 42
defaultMapBiasExpMean ............................................... 42
getSnpldFromLocation .................................................. 43
GlobalAnalysis-class .................................................... 44
GRvariants ................................................................. 45
histplot ................................................................. 45
implodeList.old ............................................................. 46
import-bam ................................................................. 47
import-bam-2 ............................................................... 48
import-bcf ................................................................. 49
inferAlleles ............................................................... 50
AllelicImbalance-package

A package meant to provide all basic functions for high-throughput allele specific expression analysis

Description

Package AllelicImbalance has functions for importing, filtering and plotting high-throughput data to make an allele specific expression analysis. A major aim of this package is to provide functions to collect as much information as possible from regions of choice, and to be able to explore the allelic expression of that region in detail.

Details

| Package:   | AllelicImbalance |
| Type:      | Package          |
| Version:   | 1.2.0            |
| Date:      | 2014-08-24       |
| License:   | GPL-3            |
Overview - standard procedure

Start out creating a GRange object defining the region of interest. This can also be done using getAreaFromGeneNames providing gene names as arguments. Then use BamImpGAList to import reads from that region and find potential SNPs using scanForHeterozygotes. Then retrieve the allele counts of heterozygote sites by the function getAlleleCount. With this data create an ASEset. At this point all pre-requisites for a ‘basic’ allele specific expression analysis is available. Two ways to go on could be to apply chisq.test or barplot on this ASEset object.

Author(s)

Author: Jesper Robert Gadin Author: Lasse Folkersen
Maintainer: Jesper Robert Gadin <j.r.gadin@gmail.com>

References

Reference to published application note (work in progress)

See Also

• code?ASEset

annotation-wrappers AnnotationDb wrappers

Description

These functions acts as wrappers to retrieve information from annotation database objects (annotationDb objects) or (transcriptDb objects)

Usage

getGenesFromAnnotation(OrgDb, GR, leftFlank = 0, rightFlank = 0, getUCSC = FALSE, verbose = FALSE)
getGenesVector(OrgDb, GR, leftFlank = 0, rightFlank = 0, verbose = FALSE)
getExonsFromAnnotation(TxDb, GR, leftFlank = 0, rightFlank = 0, verbose = FALSE)
getExonsVector(TxDb, GR, leftFlank = 0, rightFlank = 0, verbose = FALSE)
getTranscriptsFromAnnotation(TxDb, GR, leftFlank = 0, rightFlank = 0, verbose = FALSE)
getTranscriptsVector(TxDb, GR, leftFlank = 0, rightFlank = 0, verbose = FALSE)
getCDSFromAnnotation(TxDb, GR, leftFlank = 0, rightFlank = 0, verbose = FALSE)
getCDSVector(TxDb, GR, leftFlank = 0, rightFlank = 0, verbose = FALSE)
getAnnotationDataFrame(GR, strand = "+", annotationType = NULL, OrgDb = NULL, TxDb = NULL, verbose = FALSE)

Arguments

OrgDb: An OrgDb object
GR: A GenomicRanges object with sample area
leftFlank: An integer specifying number of additional nucleotides around the SNPs for the leftFlank
rightFlank: An integer specifying number of additional nucleotides around the SNPs for the rightFlank
getUCSC: A logical indicating if UCSC transcript IDs should also be retrieved
verbose: A logical making the functions more talkative
TxDb: A transcriptDb object
strand: Two options, ‘+’ or ‘-’
annotationType: select one or more from ‘gene’, ‘exon’, ‘transcript’, ‘cds’.

Details

These functions retrieve regional annotation from OrgDb or TxDb objects, when given GRanges objects.

Value

GRanges object with ranges over the genes in the region.
The getGenesVector function will return a character vector where each element are gene symbols separated by comma
GRanges object with ranges over the exons in the region.
The getExonsVector function will return a character vector where each element are exons separated by comma
GRanges object with ranges over the CDSs in the region.
The getCDSVector function will return a character vector where each element are CDSs separated by comma
The getAnnotationDataFrame function will return a data.frame with annotations. This function is used internally by i.e. the barplot-function

Author(s)

Jesper R. Gadin, Lasse Folkersen
Examples

```r
data(ASEset)
require(org.Hs.eg.db)
require(TxDb.Hsapiens.UCSC.hg19.knownGene)
OrgDb <- org.Hs.eg.db
TxDb <- TxDb.Hsapiens.UCSC.hg19.knownGene

# use for example BcfFiles as the source for SNPs of interest
GR <- rowRanges(ASEset)
# get annotation
g <- getGenesFromAnnotation(OrgDb,GR)
e <- getExonsFromAnnotation(TxDb,GR)
t <- getTranscriptsFromAnnotation(TxDb,GR)
c <- getCDSFromAnnotation(TxDb,GR)
```

---

**annotationBarplot**  
*add annotation to AllelicImbalance barplot*

**Description**

adds a customizable annotation functionality for AllelicImbalance barplots.

**Usage**

```r
annotationBarplot(strand, snp, lowerLeftCorner, annDfPlus, annDfMinus, cex = 0.7, ypos = 0, interspace = 1)
```

**Arguments**

- **strand**: strand. "+", ",", "*" or "both"
- **snp**: integer for the described snp
- **lowerLeftCorner**: position of the plot to add legend to (default c(0,0))
- **annDfPlus**: annotation dataframe plus strand
- **annDfMinus**: annotation dataframe minus strand
- **cex**: size of annotation text
- **ypos**: relative y-axis position for the annotation text
- **interspace**: space between each annotation block

**Details**

the function is preferably called from within the AllelicImbalance barplot method.

**Author(s)**

Jesper R. Gadin
Examples

# code placeholders
# < create a barplot without annotation >
# < add annotation >

## ASEset-barplot

### Description

Generates barplots for ASEset objects. Two levels of plotting detail are provided: a detailed barplot of read counts by allele useful for fewer samples and SNPs, and a less detailed barplot of the fraction of imbalance, useful for more samples and SNPs.

### Usage

barplot(height, ...)  

## S4 method for signature 'ASEset'
barplot(height, type = "count", sampleColour.top = NULL, sampleColour.bot = NULL, legend = TRUE, pValue = TRUE, strand = "*", testValue = NULL, testValue2 = NULL, OrgDb = NULL, TxDb = NULL, annotationType = c("gene", "exon", "transcript"), main = NULL, ylim = NULL, yaxis = TRUE, xaxis = FALSE, ylab = TRUE, ylab.text = NULL, xlab.text = "samples", xlab = TRUE, legend.colnames = "", las.ylab = 1, las.xlab = 2, cex.main = 1, cex.pValue = 0.7, cex.ylab = 0.7, cex.xlab = 0.7, cex.legend = 0.6, add = FALSE, lowerLeftCorner = c(0, 0), size = c(1, 1), addHorizontalLine = 0.5, add.frame = TRUE, filter.pValue.fraction = 0.99, legend.fill.size = 1, legend.interspace = 1, verbose = FALSE, top.fraction.criteria = "maxcount", cex.annotation = 0.7, ypos.annotation = 0, annotation.interspace = 1, ...)

### Arguments

- **height**: An ASEset object  
- **...**: for simpler generics when extending function  
- **type**: 'count' or 'fraction'  
- **sampleColour.top**: User specified colours for top fraction  
- **sampleColour.bot**: User specified colours for bottom fraction  
- **legend**: Display legend  
- **pValue**: Display p-value  
- **strand**: four options, '+', '-', 'both' or '*'  
- **testValue**: if set, a matrix or vector with user p-values
testValue2 if set, a matrix or vector with user p-values
OrgDb an OrgDb object which provides annotation
TxDb a TxDb object which provides annotation
annotationType select one or more from 'gene', 'exon', 'transcript', 'cds'.
main text to use as main label
ylim set plot y-axis limit
yaxis wheter the y-axis is to be displayed or not
xaxis wheter the x-axis is to be displayed or not
ylab showing labels for the tic marks
ylab.text ylab text
xlab.text xlab text
xlab showing labels for the tic marks
legend.colnames gives colnames to the legend matrix
las.ylab orientation of ylab text
las.xlab orientation of xlab text
cex.main set main label size (max 2)
cex.pValue set pValue label size
cex.ylab set ylab label size
cex.xlab set xlab label size
cex.legend set legend label size
add boolean indicates if a new device should be started
lowerLeftCorner integer that is only useful when add=TRUE
size Used internally by locationplot. Rescales each small barplot window
addHorizontalLine adds a horizontal line that marks the default fraction of 0.5 - 0.5
add.frame boolean to give the new plot a frame or not
filter.pValue.fraction numeric between 0 and 1 that filter away pValues where the main allele has this frequency.
legend.fill.size size of the fill/boxes in the legend (default:NULL)
legend.interspace set legend space between fills and text
verbose Makes function more talkative
top.fraction.criteria 'maxcount', 'ref' or 'phase'
cex.annotation size of annotation text
ypos.annotation relative ypos for annotation text
annotation.interspace space between annotation text
Details

filter.pValue.fraction is intended to remove p-value annotation with very large difference in frequency, which could just be a sequencing mistake. This is to avoid p-values like 1e-235 or similar.

sampleColourUser specified colours, either given as named colours ('red', 'blue', etc) or as hexadecimal code. Can be either length 1 for all samples, or else of a length corresponding to the number of samples for individual colouring.

Author(s)

Jesper R. Gadin, Lasse Folkersen

See Also

• The ASEset class which the barplot function can be called up on.

Examples

data(ASEset)
barplot(ASEset[1])

Description

Object that holds allele counts, genomic positions and map-bias for a set of SNPs

Usage

alleleCounts(x, strand = "*", return.class = "list")

## S4 method for signature 'ASEset'
alleleCounts(x, strand = "*", return.class = "list")

alleleCounts(x, ...) <- value

## S4 replacement method for signature 'ASEset'
alleleCounts(x, strand = "*", return.class = "array", ...

mapBias(x, ...)

## S4 method for signature 'ASEset'
mapBias(x, return.class = "list")

fraction(x, ...)

## S4 method for signature 'ASEset'
fraction(x, strand = "*", 
    top.fraction.criterion = "maxcount", verbose = FALSE, ...)

arank(x, return.type = "names", return.class = "list", strand = "*", ...)

frequency(x, ...)

genotype(x, ...)

## S4 method for signature 'ASEset'
genotype(x, return.class = "matrix")

genotype(x) <- value

## S4 replacement method for signature 'ASEset'
genotype(x) <- value

countsPerSnp(x, ...)

## S4 method for signature 'ASEset'
countsPerSnp(x, return.class = "matrix", 
    return.type = "mean", strand = "*")

countsPerSample(x, ...)

## S4 method for signature 'ASEset'
countsPerSample(x, return.class = "matrix", 
    return.type = "mean", strand = "*")

phase(x, ...)

## S4 method for signature 'ASEset'
phase(x, return.class = "matrix")

phase(x) <- value

## S4 replacement method for signature 'ASEset'
phase(x) <- value

mapBias(x) <- value

## S4 replacement method for signature 'ASEset'
mapBias(x) <- value

refExist(x)

## S4 method for signature 'ASEset'
refExist(x)

ref(x)

## S4 method for signature 'ASEset'
ASEset-class

ref(x)
ref(x) <- value

## S4 replacement method for signature 'ASEset,ANY'
ref(x) <- value

altExist(x)

## S4 method for signature 'ASEset'
altExist(x)

alt(x)

## S4 method for signature 'ASEset'
alt(x)
alt(x) <- value

## S4 replacement method for signature 'ASEset,ANY'
alt(x) <- value

aquals(x, ...)

## S4 method for signature 'ASEset'
aquals(x)
aquals(x) <- value

## S4 replacement method for signature 'ASEset'
aquals(x) <- value

maternalAllele(x, ...)

## S4 method for signature 'ASEset'
maternalAllele(x)

paternalAllele(x, ...)

## S4 method for signature 'ASEset'
paternalAllele(x)

Arguments

x \hspace{1cm} \text{ASEset object}
strand \hspace{1cm} \text{which strand of ‘+’, ‘-’ or ‘*’}
return.class \hspace{1cm} \text{return ‘list’ or ‘array’}
... \hspace{1cm} \text{additional arguments}
value \hspace{1cm} \text{replacement variable}
top.fraction.criteria
\hspace{1cm} \text{‘maxcount’, ‘ref’ or ‘phase’}
 verbose makes function more talkative  
 return.type return 'names', rank or 'counts'

Details

An ASEset object differs from a regular RangedSummarizedExperiment object in that the assays contains an array instead of matrix. This array has ranges on the rows, sampleNames on the columns and variants in the third dimension.

It is possible to use the commands barplot and locationplot on an ASEset object see more details in barplot and locationplot.

Three different alleleCount options are available. The simplest one is the * option, and is for experiments where the strand information is not known e.g. non-stranded data. The unknown strand could also be for strand specific data when the aligner could not find any strand associated with the read, but this should normally not happen, and if it does probably having an extremely low mapping quality. Then there are an option too add plus and minus stranded data. When using this, it is essential to make sure that the RNA-seq experiment under analysis has in fact been created so that correct strand information was obtained. The most functions will by default have their strand argument set to '*'.

The phase information is stored by the convention of `maternal chromosome|paternal chromosome`, with 0 as reference allele and 1 as alternative allele. '/' when the phase is known and '|/' when the phase is unknown. Internally the information will be stored as an three dimensional array, dim 1 for SNPs, dim 2 for Samples and dim 3 which is fixed and stores maternal chromosome, paternal chromosome and phased (1 equals TRUE).

Value

An object of class ASEset containing location information and allele counts for a number of SNPs measured in a number of samples on various strand, as well as mapBias information. All data is stored in a manner similar to the RangedSummarizedExperiment class.

Table

table(...)  
Arguments:
... An ASEset object that contains the variants of interest  
The generics for table does not easily allow more than one argument so in respect to the different strand options, table will return a SimpleList with length 3, one element for each strand.

Frequency

frequency(x, return.class = "list", strand = "*", threshold.count.sample = 15)  
Arguments:
x An ASEset object that contains the variants of interest  
x threshold.count.samples if sample has fewer counts the function return NA.

Constructor

ASEsetFromCountList(rowRanges, countListNonStranded = NULL, countListPlus = NULL, countListMinus = NULL, countListUnknown = NULL, colData = NULL, mapBiasExpMean = array(), verbose = FALSE, ...)
Arguments:
ASEset-class

rowRanges  A GenomicRanges object that contains the variants of interest
countListNonStranded  A list where each entry is a matrix with allele counts as columns and sample counts as rows
countListPlus  A list where each entry is a matrix with allele counts as columns and sample counts as rows
countListMinus  A list where each entry is a matrix with allele counts as columns and sample counts as rows
countListUnknown  A list where each entry is a matrix with allele counts as columns and sample counts as rows
colData  A DataFrame object containing sample specific data
mapBiasExpMean  A 3D array describing mapping bias. The SNPs are in the 1st dimension, samples in the 2nd dimension and variants in the 3rd dimension.
verbose  Makes function more talkative
... arguments passed on to SummarizedExperiment constructor

Author(s)
Jesper R. Gadin, Lasse Folkersen

See Also
- RangedSummarizedExperiment objects.

Examples

```r
#make example countList
set.seed(42)
countListPlus <- list()
snps <- c('snp1','snp2','snp3','snp4','snp5')
for(snp in snps){
  count<-matrix(rep(0,16),ncol=4,dimnames=list(
    c('sample1','sample2','sample3','sample4'),
    c('A','T','G','C')))
  #insert random counts in two of the alleles
  for(allele in sample(c('A','T','G','C'),2)){
    count[,allele]<-as.integer(rnorm(4,mean=50,sd=10))
  }
  countListPlus[[snp]] <- count
}

#make example rowRanges
rowRanges <- GRanges(
  seqnames = Rle(c('chr1', 'chr2', 'chr1', 'chr3', 'chr1')),
  ranges = IRanges(1:5, width = 1, names = head(letters,5)),
  snp = paste('snp',1:5,sep='')
)

#make example colData
colData <- DataFrame(Treatment=c('ChIP', 'Input', 'Input', 'ChIP'),
  row.names=c('ind1','ind2','ind3','ind4'))
```
#make ASEset
a <- ASEsetFromCountList(rowRanges, countListPlus=countListPlus, colData=colData)

#example phase matrix (simple form)
p1 <- matrix(sample(c(1,0),replace=TRUE, size=nrow(a)*ncol(a)),nrow=nrow(a), ncol(a))
p2 <- matrix(sample(c(1,0),replace=TRUE, size=nrow(a)*ncol(a)),nrow=nrow(a), ncol(a))
p <- matrix(paste(p1,sample(c("|","|","/"), size=nrow(a)*ncol(a), replace=TRUE), p2, sep=""), nrow=nrow(a), ncol(a))

phase(a) <- p

#generate ASEset from array
snps <- 999
samples <-5
ar <-array(rep(unlist(lapply(1:snps, function(x){(sample(c(TRUE,FALSE,TRUE,FALSE), size = 4))})), samples), dim=c(4,snps,samples))
ar2 <- array(sample(50:300, 4*snps*samples,replace=TRUE), dim=c(4,snps,samples))
ar2[ar] <- 0
ar2 <- aperm(ar2, c(2, 3, 1))
dimnames(ar2) <- list(paste("snp",1:snps,sep=""),paste("sample",1:samples,sep=""), c("A","C","G","T"))
gr <- GRanges(seqnames=c("chr2"), ranges=IRanges(start=1:dim(ar2)[1], width=1), strand="*")
a <- ASEsetFromArrays(gr, countsUnknown=ar2)

ASEset-filters               genotype filter methods

**Description**

useful genotype filters

**Usage**

hetFilt(x, ...)

## S4 method for signature 'ASEset'
hetFilt(x, source = "genotype", ...)

**Arguments**

- x: ASEset object

- ...: internal param

- source: ‘genotype’ or ‘alleleCounts’

**Details**

hetFilt returns TRUE if the samples is heterozygote, based on stored genotype information present in the phase data.
Author(s)
Jesper R. Gadin, Lasse Folkersen

Examples

```
#load example data
data(ASEset)
a <- ASEset

genotype(a) <- inferGenotypes(a)
hets <- hetFilt(a)
```

Description
Generates gbarplots for ASEset objects. Two levels of plotting detail are provided: a detailed gbarplot of read counts by allele useful for fewer samples and SNPs, and a less detailed gbarplot of the fraction of imbalance, useful for more samples and SNPs.

Usage

```
gbarplot(x, type = "count", strand = "*", verbose = FALSE, ...)
```

Arguments

- `x`: An ASEset object
- `type`: 'count' or 'fraction'
- `strand`: four options, '+', '-', 'both' or '*'
- `verbose`: Makes function more talkative
- `...`: for simpler generics when extending function

Details
This function serves the same purpose as the normal barplot, but with trellis graphics using lattice, to be able to integrate well with Gviz track functionality.

Author(s)
Jesper R. Gadin

See Also
- The ASEset class which the gbarplot function can be called up on.
- The barplot non trellis barplot.
Examples

data(ASEset)
gbarplot(ASEset[1])

Description
plotting ASE effects over a specific genomic region using Gviz functionality

Usage

glocationplot(x, type = "fraction", strand = "*", BamGAL = NULL,
    GenomeAxisTrack = FALSE, trackNameDeAn = paste("deTrack", type),
    TxDb = NULL, sizes = NULL, add = FALSE, verbose = FALSE, ...)

Arguments

x an ASEset object.
type 'fraction' or 'count'
strand '+','-','*' or 'both'. This argument determines which strand is plotted. See getAlleleCounts for more information of choice of strand.
BamGAL GAlignmentsList covering the same genomic region as the ASEset
GenomeAxisTrack include an genomic axis track
trackNameDeAn trackname for deAnnotation track
TxDb a TxDb object which provides annotation
sizes vector with the sum 1. Describes the size of the tracks
add add to existing plot
verbose if set to TRUE it makes function more talkative
... arguments passed on to barplot function

Details
The glocationplot methods visualises the distribution of ASE over a larger region on one chromosome. It takes and ASEset object as well as additional information on plot type (see gbarplot), strand type (see getAlleleCounts), Annotation tracks are created from the Gviz packageh. It is obviously important to make sure that the genome build used is set correctly, e.g. 'hg19'.
sizes has to be of the same length as the number of tracks used.

Author(s)
Jesper R. Gadin
See Also

- The ASEset class which the glocationplot function can be called up on.

Examples

```r
data(ASEset)
genome(ASEset) <- 'hg19'
glocationplot(ASEset,strand='+')

# for ASEsets with fewer SNPs the 'count' type plot is useful
glocationplot(ASEset,type='count',strand='+')
```

---

**ASEset-gviztrack**

**ASEset-gviztrack ASESet objects**

### Description

plotting ASE effects over a specific genomic region

### Usage

```r
ASEDAnnotationTrack(x, GR = rowRanges(x), type = "fraction", strand = "*",
                      trackName = paste("deTrack", type), verbose = TRUE, ...)
```

**## S4 method for signature 'ASEset'**

```r
ASEDAnnotationTrack(x, GR = rowRanges(x),
                      type = "fraction", strand = "+", trackName = paste("deTrack", type),
                      verbose = TRUE, ...)
```

```r
CoverageDataTrack(x, GR = rowRanges(x), BamList = NULL, strand = NULL,
                   start = NULL, end = NULL, trackNameVec = NULL, meanCoverage = FALSE,
                   verbose = TRUE, ...)
```

### Arguments

- **x**
  - an ASEset object.
- **GR**
  - genomic range of plotting
- **type**
  - 'fraction' or 'count'
- **strand**
  - '+','-'. This argument determines which strand is plotted.
- **trackName**
  - name of track (ASEDAnnotationTrack)
- **verbose**
  - Setting verbose=TRUE gives details of procedure during function run
- **BamList**
  - GAAlignmentsList object of reads from the same genomic region as the ASEset
- **start**
  - start position of reads to be plotted
- **end**
  - end position of reads to be plotted
- **trackNameVec**
  - names of tracks (CoverageDataTrack)
- **meanCoverage**
  - mean of coverage over samples (CoverageGataTrack)
Details
For information of how to use these tracks in more ways, visit the Gviz package manual.

Author(s)
Jesper R. Gadin

See Also
• The ASEset class which the functions can be called up on.

Examples

```r
data(ASEset)
set.seed(1)
x <- ASEset[,1:2]
r <- reads[1:2]
genome(x) <- 'hg19'
seqlevels(r) <- seqlevels(x)

GR <- GRanges(seqnames=seqlevels(x),
ranges=IRanges(start=min(start(x)),end=max(end(x))),
strand=('+', genome=genome(x))

detTrack <- ASEDAnnotationTrack(x, GR=GR, type='fraction', strand='+' )
covTracks <- CoverageDataTrack(x,BamList=r,strand=')+

lst <- c(detTrack,covTracks)
sizes <- c(0.5, rep(0.5/length(covTracks),length(covTracks)))

plotTracks(lst, from=min(start(x)), to=max(end(x)),
sizes=sizes, col.line = NULL, showId = FALSE, main='mainText',
cex.main=1, title.width=1, type='histogram')
```

ASEset-locationplot  locationplot ASEset objects

Description
plotting ASE effects over a specific genomic region

Usage
locationplot(x, ...)

# S4 method for signature 'ASEset'
locationplot(x, type = "fraction", strand = "+",
         yaxis = TRUE, xaxis = FALSE, xlab = FALSE, ylab = TRUE,
         xlab.text = "", ylab.text = "", legend.colnames = ", size = c(0.8, 1),
         main = NULL, pValue = FALSE, cex.main = 0.7, cex.ylab = 0.6,
cex.legend = 0.5, OrgDb = NULL, TxDb = NULL, verbose = TRUE,
top.fraction.criteria = "maxcount", allow.whole.chromosome = FALSE, ...)

Arguments

x an ASEset object.
...
arguments passed on to barplot function
type 'fraction' or 'count'
strand '+'; '-'; '*'; or 'both'. This argument determines which strand is plotted. See getAlleleCounts for more information on strand.
yaxis wheter the y-axis is to be displayed or not
xaxis wheter the x-axis is to be displayed or not
xlab showing labels for the tic marks
ylab showing labels for the tic marks
xlab.text xlab text
ylab.text ylab text
legend.colnames gives colnames to the legend matrix
size will give extra space in the margins of the inner plots
main text to use as main label
pValue Display p-value
cex.main set main label size
cex.ylab set ylab label size
cex.legend set legend label size
OrgDb an OrgDb object from which to plot a gene map. If given together with argument
TxDb this will only be used to extract genesymbols.
TxDb a TxDb object from which to plot an exon map.
verbose Setting verbose=TRUE gives details of procedure during function run
top.fraction.criteria
  'maxcount', 'ref' or 'phase'
allow.whole.chromosome logical, overrides 200kb region limit, defaults to FALSE

Details

The locationplot methods visualises how fractions are distributed over a larger region of genes on
one chromosome. It takes and ASEset object as well as additional information on plot type (see barplot), strand type (see getAlleleCounts), colouring, as well as annotation. The annotation is
taken either from the bioconductor OrgDb-sets, the TxDb sets or both. It is obviously important to
make sure that the genome build used is the same as used in aligning the RNA-seq data.

Author(s)

Jesper R. Gadin, Lasse Folkersen

See Also

• The ASEset class which the locationplot function can be called up on.
Examples

data(ASEset)
locationplot(ASEset)

#SNPs are plotted in the order in which they are found.
#This can be sorted according to location as follows:
locationplot(ASEset[order(start(rowRanges(ASEset))),])

#for ASEsets with fewer SNPs the 'count' type plot is
# useful for detailed visualization.
locationplot(ASEset,type='count',strand='*')

Description

Identifies the positions of SNPs found in BamGR reads.

Usage

scanForHeterozygotes(BamList, ...)

## S4 method for signature 'GAlignmentsList'
scanForHeterozygotes(BamList,
minimumReadsAtPos = 20, maximumMajorAlleleFrequency = 0.9,
minimumMinorAlleleFrequency = 0.1, minimumBiAllelicFrequency = 0.9,
verbose = TRUE, ...)

Arguments

BamList A GAlignmentsList object
... argument to pass on
minimumReadsAtPos minimum number of reads required to call a SNP at a given position
maximumMajorAlleleFrequency maximum frequency allowed for the most common allele. Setting this parameter lower will minimise the SNP calls resulting from technical read errors, at the cost of missing loci with potential strong ASE
minimumMinorAlleleFrequency minimum frequency allowed for the second most common allele. Setting this parameter higher will minimise the SNP calls resulting from technical read errors, at the cost of missing loci with potential strong ASE
minimumBiAllelicFrequency minimum frequency allowed for the first and second most common allele. Setting a Lower value for this parameter will minimise the identification of loci with three or more alleles in one sample. This is useful if sequencing errors are suspected to be common.
**Verbose**

logical indicating if process information should be displayed

**Details**

This function scans all reads stored in a `GAlignmentsList` for possible heterozygote positions. The user can balance the sensitivity of the search by modifying the `minimumReadsAtPos`, `maximumMajorAlleleFrequency` and `minimumBiAllelicFrequency` arguments.

**Value**

`scanForHeterozygotes` returns a `GRanges` object with the SNPs for the `BamList` object that was used as input.

**Author(s)**

Jesper R. Gadin, Lasse Folkersen

**See Also**

- The `getAlleleCounts` which is a function that count the number of reads overlapping a site.

**Examples**

```r
data(reads)
s <- scanForHeterozygotes(reads, verbose=FALSE)
```

---

**ASEset.old**

**ASEset.old object**

**Description**

old version of an ASEset which needs to be updated

**Author(s)**

Jesper R. Gadin, Lasse Folkersen

**Examples**

```r
## load example data (Not Run)
data(ASEset.old)
```
ASEset.sim

**ASEset.sim object**

**Description**

ASEset with simulated data with SNPs within the first 200bp of chromosome 17, which is required to have example data for the refAllele function.

**Author(s)**

Jesper R. Gadin, Lasse Folkersen

**Examples**

```r
##load example data (Not Run)
data(ASEset.sim)
```

ASEsetFromBam

**ASEset from bam file**

**Description**

count alleles and create an ASEset direct from bam file instead of reading into R first.

**Usage**

```r
ASEsetFromBam(gr, ...)
```

```r
## S4 method for signature 'GRanges'
ASEsetFromBam(gr, pathToDir, PE = TRUE,
               flagsMinusStrand = c(83, 163), flagsPlusStrand = c(99, 147),
               strandUnknown = FALSE, ...)
```

**Arguments**

- `gr` GenomicRanges of SNPs to create ASEset for
- `...` passed on to ASEsetFromBam function
- `pathToDir` Directory of bam files with index in same directory
- `PE` if paired end or not (default: TRUE)
- `flagsMinusStrand` flags that mark reads coming from minus strand
- `flagsPlusStrand` flags that mark reads coming from plus strand
- `strandUnknown` default: FALSE
Details

counts the alleles in a bam file based on GRanges positions.

Author(s)

Jesper R. Gadin

Examples

data(GRvariants)
gr <- GRvariants

##no execution at the moment
#pathToDir <- system.file('inst/extdata/ERP000101_subset', package='AllelicImbalance')
#a <- ASEsetFromBam(gr, pathToDir)

Description

Generates lattice barplots for ASEset objects. Two levels of plotting detail are provided: a detailed barplot of read counts by allele useful for fewer samples and SNPs, and a less detailed barplot of the fraction of imbalance, useful for more samples and SNPs.

Usage

barplotLatticeFraction(identifier, ...)
barplotLatticeCounts(identifier, ...)

Arguments

identifier, the single snp name to plot
... used to pass on variables

Details

filter.pValue.fraction is intended to remove p-value annotation with very large difference in frequency, which could just be a sequencing mistake. This is to avoid p-values like 1e-235 or similar.

sampleColourUser specified colours, either given as named colours ('red', 'blue', etc) or as hexadecimal code. Can be either length 1 for all samples, or else of a length corresponding to the number of samples for individual colouring.

Author(s)

Jesper R. Gadin, Lasse Folkersen
See Also

- The ASEset class which the barplot function can be called up on.

Examples

```r
a <- ASEset
name <- rownames(a)[1]

barplotLatticeFraction(identifier=name, x=a, astrand="+")
barplotLatticeCounts(identifier=name, x=a, astrand="+")
```

---

**Description**

Performs a binomial test on an ASEset object.

**Usage**

```r
## S4 method for signature 'ASEset'
binom.test(x, n = "*")
```

**Arguments**

- `x` ASEset object
- `n` strand option

**Details**

the test can only be applied to one strand at the time.

**Value**

- `binom.test` returns a matrix

**Author(s)**

Jesper R. Gadin, Lasse Folkersen

**See Also**

- The chisq.test which is another test that can be applied on an ASEset object.
chisq.test

Examples

# load example data
data(ASEset)

# make a binomial test
binom.test(ASEset,'*')

chisq.test              chi-square test

Description

Performs a chisq.test on an ASEset object.

Usage

## S4 method for signature 'ASEset'
chisq.test(x, y = "*")

Arguments

x  ASEset object
y  strand option

Details

The test is performed on one strand in an ASEset object.

Value

chisq.test returns a matrix with the chisq.test P-value for each SNP and sample

Author(s)

Jesper R. Gadin, Lasse Folkersen

See Also

- The binom.test which is another test that can be applied on an ASEset object.

Examples

# load example data
data(ASEset)

# make a chi-square test on default non-stranded strand
chisq.test(ASEset)
**countAllelesFromBam**

**Description**

count alleles before creating ASEse.

**Usage**

```
countAllelesFromBam(gr, ...)  
```  

## S4 method for signature 'GRanges'

```
countAllelesFromBam(gr, pathToDir, flag = NULL, 
                    scanBamFlag = NULL, return.class = "array", verbose = TRUE, ...)  
```  

**Arguments**

- `gr` : GRanges that contains SNPs of interest
- `...` : arguments to pass on
- `pathToDir` : path to directory of bam files
- `flag` : specify one flag to use as filter, default is no filtering. allowed flags are 99, 147, 83 and 163
- `scanBamFlag` : set a custom flag to use as filter
- `return.class` : type of class for the returned object
- `verbose` : makes function more talkative

**Details**

counts the alleles in a bam file based on GRanges positions.

Important excerpt from the details section of the internal applyPileups function: Regardless of 'param' values, the algorithm follows samtools by excluding reads flagged as unmapped, secondary, duplicate, or failing quality control.

**Author(s)**

Jesper R. Gadin

**Examples**

data(GRvariants)  
gr <- GRvariants

```r  
##not run at the moment  
#pathToDir <- system.file('inst/extdata/ERP000101_subset', package=' AllelicImbalance')  
#ar <- countAllelesFromBam(gr, pathToDir)  
```
coverageMatrixListFromGAL

description of GAlignmentsList

Description

Get coverage per nucleotide for reads covering a region

Usage

coverageMatrixListFromGAL(BamList, ...)

## S4 method for signature 'GAlignmentsList'
coverageMatrixListFromGAL(BamList, strand = "*",
ignore.empty.bam.row = TRUE)

Arguments

BamList       GAlignmentsList containing reads over the region to calculate coverage
...           arguments to pass on
strand        strand has to be '+' or '-'
ignore.empty.bam.row
argument not in use atm

Details

a convenience function to get the coverage from a list of reads stored in GAlignmentsList, and returns by default a list with one matrix, and information about the genomic start and stop positions.

Author(s)

Jesper R. Gadin

Examples

r <- reads
seqlevels(r) <- '17'
covMatList <- coverageMatrixListFromGAL(BamList=r, strand='+')
defaultMapBias  
*Generate default mapbias from genotype*

**Description**

Create mapbias array from genotype matrix requires genotype information

**Usage**

```r
defaultMapBias(x, ...)  
## S4 method for signature 'ASEset'  
defaultMapBias(x, return.class = "array")
```

**Arguments**

- `x`: ASEset object
- `...`: internal arguments
- `return.class`: "array" or "ASEset"

**Details**

Default mapbias will be 0.5 for bi-allelic snps and 1 for homozygots. For genotypes with NA, 0.5 will be placed on all four alleles. Therefore tri-allelic can not be used atm. Genotype information has to be placed in the genotype(x) assay.

**Author(s)**

Jesper R. Gadin, Lasse Folkersen

**Examples**

```r
#load example data  
data(ASEset.sim)  
fasta <- system.file("extdata/hg19.chr17.subset.fa", package="AllelicImbalance")  
refAllele(ASEset.sim,fasta=fasta)  
a <- refAllele(ASEset.sim,fasta=fasta)
```
**defaultPhase**

**Description**

used to populate the phase slot in an ASEset object

**Usage**

```r
defaultPhase(i, ...)  
```

```r   ## S4 method for signature 'numeric'   
defaultPhase(i, j, ...)```

**Arguments**

- `i` number of rows
- `j` number of columns
- `...` arguments to forward to internal functions

**Details**

will set everything to 0

**Author(s)**

Jesper R. Gadin, Lasse Folkersen

**Examples**

```r
i <- 5  
j <- 10  
defaultPhase(i,j)```

---

**detectAI**

**Description**

detection of AllelicImbalance
Usage
detectAI(x, ...)

## S4 method for signature 'ASEset'
detectAI(x, return.class = "DetectedAI", strand = "x",
threshold.frequency = 0, threshold.count.sample = 1,
threshold.delta.frequency = 0, threshold.pvalue = 0.05,
inferGenotype = FALSE, random.ref = FALSE, function.test = "binom.test",
verbose = TRUE, gc = FALSE, biasMatrix = FALSE)

Arguments
x ASEset
...
internal arguments
return.class class to return (atm only class 'logical')
strand strand to infer from
threshold.frequency least fraction to classify (see details)
threshold.count.sample least amount of counts to try to infer allele
threshold.delta.frequency minimum of frequency difference from 0.5 (or mapbias adjusted value)
threshold.pvalue pvalue over this number will be filtered out
inferGenotype infer genotypes based on count data in ASEset object
random.ref set the reference as random if you dont know. Affects interpretation of results.
function.test At the moment the only available option is 'binomial.test'
verbose makes function more talkative
gc use garbage collection when possible to save space
biasMatrix use biasMatrix in ASEset, or use default expected frequency of 0.5 for all sites

Details
threshold.frequency is the least fraction needed to classify as bi tri or quad allelic SNPs. If 'all'
then all of bi tri and quad allelic SNPs will use the same threshold. Everything under the threshold
will be regarded as noise. 'all' will return a matrix with snps as rows and uni bi tri and quad will be
columns. For this function Anything that will return TRUE for tri-allelic will also return TRUE for
uni and bi-allelic for the same SNP an Sample.
return.type 'ref' return only AI when reference allele is more expressed. 'alt' return only AI when
alternative allele is more expressed or 'all' for both 'ref' and 'alt' alleles. Reference allele is the
one present in the reference genome on the forward strand.
threshold.delta.frequency and function.test will use the value in mapBias(x) as expected value.
function.test will use the two most expressed alleles for testing. Make therefore sure there are no
tri-allelic SNPs or somatic mutations among the SNPs in the ASEset.
inferGenotype(), set TRUE it should be used with as much samples as possible. If you split up
the samples and run detectAI() on each sample separately, please make sure you have inferred the
genotypes in before hand, alternatively used the genotypes detected by another variantCaller or
chip-genotypes. Use ONLY biallelic genotypes.
DetectedAI-class

Author(s)
Jesper R. Gadin

Examples

# load example data
data(ASEset)
a <- ASEset

dai <- detectAI(a)

DetectedAI-class

DetectedAI class

Description

Object that holds results from AI detection.

Usage

referenceFrequency(x, ...)

## S4 method for signature 'DetectedAI'
referenceFrequency(x, return.class = "array")

thresholdFrequency(x, ...)

## S4 method for signature 'DetectedAI'
thresholdFrequency(x, return.class = "array")

thresholdCountSample(x, ...)

## S4 method for signature 'DetectedAI'
thresholdCountSample(x, return.class = "array")

thresholdDeltaFrequency(x, ...)

## S4 method for signature 'DetectedAI'
thresholdDeltaFrequency(x, return.class = "array")

thresholdPvalue(x, ...)

## S4 method for signature 'DetectedAI'
thresholdPvalue(x, return.class = "array")

Arguments

x             ASEset object or list of ASEsets
...           pass arguments to internal functions
return.class  type of class returned eg. "list" or "array".
Details

The DetectedAI-class contains

Author(s)

Jesper R. Gadin, Lasse Folkersen

Examples

data(ASEset)
a <- ASEset
dai <- detectAI(a)

#summary(gba)
#write.tables(dai)

Description

plot functions for the DetectedAI-class

Usage

frequency_vs_threshold_variable_plot(x, ...)

## S4 method for signature 'DetectedAI'
frequency_vs_threshold_variable_plot(x,
  var = "threshold.count.sample", hetOverlay = TRUE,
  smoothscatter = FALSE)

detectedAI_vs_threshold_variable_plot(x, ...)

## S4 method for signature 'DetectedAI'
detectedAI_vs_threshold_variable_plot(x,
  var = "threshold.count.sample", summaryOverSamples = "sum",
  hetOverlay = TRUE, smoothscatter = FALSE)

reference_frequency_density_vs_threshold_variable_plot(x, ...)

## S4 method for signature 'DetectedAI'
reference_frequency_density_vs_threshold_variable_plot(x,
  var = "threshold.count.sample")

detectedAI_vs_threshold_variable_multigraph_plot(x, ...)

## S4 method for signature 'DetectedAI'
detectedAI_vs_threshold_variable_multigraph_plot(x, 
  ncol = 2, ...)

frequency_vs_threshold_variable_multigraph_plot(x, ...)

## S4 method for signature 'DetectedAI'
frequency_vs_threshold_variable_multigraph_plot(x, 
  ncol = 2, ...)

reference_frequency_density_vs_threshold_variable_multigraph_plot(x, ...)

## S4 method for signature 'DetectedAI'
reference_frequency_density_vs_threshold_variable_multigraph_plot(x, 
  ncol = 2, ...)

Arguments

x  
detectedAI object

...  
pass on variables internally

var  
string, see details for available options

hetOverlay  
logical, if TRUE show nr of het SNPs used to calculate the reference allele
  frequency mean

smoothscatter  
boolean, smoothscatter over the means

summaryOverSamples  
'mean' or 'sum'

ncol  
.nr of columns for multiplots

Details

plot helper functions. The documentation will be improved before next release.

Author(s)

Jesper R. Gadin, Lasse Folkersen

Examples

#some example code here
#generate example
data(ASEset)
a <- ASEset
da1 <- detectAI(a,
threshold.count.sample=1:50,
threshold.frequency=seq(0,0.5,by=0.01),
threshold.delta.frequency=seq(0,0.5,by=0.01),
threshold.pvalue=rev(seq(0.001,0.05, by=0.005))
)

frequency_vs_threshold_variable_plot(dai)
detectedAI_vs_threshold_variable_plot(dai)
detectedAI_vs_threshold_variable_multigraph_plot(dai)
Describes helper functions for the DetectedAI-class.

Usage:

- `frequency_vs_threshold_variable_summary(x, ...)

  ## S4 method for signature 'DetectedAI'
  frequency_vs_threshold_variable_summary(x,
    var = "threshold.count.sample", return.class = "matrix", ...)

- `detectedAI_vs_threshold_variable_summary(x, ...)

  ## S4 method for signature 'DetectedAI'
  detectedAI_vs_threshold_variable_summary(x,
    var = "threshold.count.sample")

- `usedSNPs_vs_threshold_variable_summary(x, ...)

  ## S4 method for signature 'DetectedAI'
  usedSNPs_vs_threshold_variable_summary(x,
    var = "threshold.count.sample")

Arguments:

- `x` detectedAI object
- `...` pass on variables internally
- `var` string, see details for available options
- `return.class` 'matrix' or 'array'

Details:

Summary helper functions. The documentation will be improved before next release.

Author(s):

Jesper R. Gadin, Lasse Folkersen
Examples

# some example code here
# generate example
data(ASEset)
a <- ASEset
dai <- detectAI(a,
threshold.count.sample=1:50,
threshold.frequency=seq(0,0.5,by=0.01),
threshold.delta.frequency=seq(0,0.5,by=0.01),
threshold.pvalue=rev(seq(0.001,0.05, by=0.005))
)
frequency_vs_threshold_variable_summary(dai)

 fractionPlotDf  

Plot Dataframe

Description
Summarizes information to ease creating plots

Usage

fractionPlotDf(x, snp, strand = ", strand = ", top.fraction.criteria = "maxcount", ...

## S4 method for signature 'ASEset'
fractionPlotDf(x, snp, strand = ", top.fraction.criteria = "maxcount", ...

Arguments

x  
ASEset

snp  
rownames identifier for ASEset or row number

strand  
"+", "-" or ":

top.fraction.criteria

'rnaxcount', 'ref' or 'phase'

...  
arguments to forward to internal functions

Details
Main purpose is to reduce the amount of overall code and ease maintenance.
top.fraction.criteria can take three options, maxcount, ref and phase. The top allele will be every second row in the data frame, with start from row 2. The maxcount argument will put the allele with most reads on top of the bivariate fraction. Similarly the ref argument will put always the reference allele on top. The phase arguments puts the maternal phase always on top. The top.fraction.criteria for the ref or phase arguments requires that both ref and alt is set in mcols(ASEset).
Author(s)

Jesper R. Gadin, Lasse Folkersen

Examples

```r
#test on example ASEset
data(ASEset)
a <- ASEset
df <- fractionPlotDf(a, 1, strand="+")
```

---

gba

global analysis wrapper

Description

A wrapper to make a global analysis based on paths for BAM, VCF and GFF files

Usage

```r
gba(pathBam, ...)
## S4 method for signature 'character'
gba(pathBam, pathVcf, pathGFF = NULL, verbose)
```

Arguments

- `pathBam`: path to bam file
- `...`: arguments to pass on
- `pathVcf`: path to vcf file
- `pathGFF`: path to gff file
- `verbose`: makes function more talkative

Author(s)

Jesper R. Gadin

Examples

```r
#empty as function doesn't exist
```
**genomatrix**  

**genomatrix object**  

**Description**  

genomatrix is an example of a matrix with genotypes

**Author(s)**  

Jesper R. Gadin, Lasse Folkersen

**Examples**

```r
#load example data (Not Run)
#data(genomatrix)
```

---

**genotype2phase**  

**genotype2phase**

**Description**

used to convert the genomatrix from the visually friendly matrix to phase array.

**Usage**

```r
genotype2phase(x, ...)  
```

```r  
## S4 method for signature 'matrix'
genotype2phase(x, ref = NULL, return.class = "array",  
levels = c("A", "C", "G", "T"), ...)  
```

**Arguments**

- `x`  
  matrix see examples
- `...`  
  pass on additional param
- `ref`  
  reference alleles
- `return.class`  
  'array' or 'list'
- `levels`  
  vector of expected alleles

**Details**

To not introduce redundant information in the ASEset object, the genotype matrix is translated to a phase matrix, containing the same information. Does not allow tri-allelic or multi-allelic SNPs, and if present the multi-allelic SNPs will lose the least occuring genotype.

This function can handle indels, but if the reference allele is not provided, the rank matrix which is temporary created might use lots of memory, depending on the amount of indels among the genotypes. As conclusion, it is preferable to send in reference genome when converting to phase.

levels information is only important if the reference allele has to be guessed, and so if reference information is provided, the levels argument can be ignored.
getAlleleCounts

**Author(s)**

Jesper R. Gadin, Lasse Folkersen

**Examples**

```r
# load example data
data(genomatrix)
data(ASEset)
p <- genotype2phase(genomatrix, ref(ASEset))
```

**Description**

Given the positions of known SNPs, this function returns allele counts from a BamGRL object.

**Usage**

```r
getAlleleCounts(BamList, ...)  
## S4 method for signature 'GAlignmentsList'
getAlleleCounts(BamList, GRvariants, strand = "*",  
    return.class = "list", verbose = TRUE, ...)
```

**Arguments**

- **BamList**: A GAlignmentsList object or GRangesList object containing data imported from a bam file.
- **...**: Parameters to pass on.
- **GRvariants**: A GRanges object that contains positions of SNPs to retrieve.
- **strand**: A length 1 character with value '+', '-', or '*'. This argument determines if getAlleleCounts will retrieve counts from all reads, or only from reads marked as '+', '-' or '*' (unknown), respectively.
- **return.class**: 'list' or 'array'.
- **verbose**: Setting verbose=TRUE makes function more talkative.

**Details**

This function is used to retrieve the allele counts from specified positions in a set of RNA-seq reads. The BamList argument will typically have been created using the impBamGAL function on bam-files. The GRvariants is either a GRanges with user-specified locations or else it is generated through scanning the same bam-files as in BamList for heterozygote locations (e.g. using scanForHeterozygotes). The GRvariants will currently only accept locations having width=1, corresponding to bi-allelic SNPs. In the strand argument, specifying '*' is the same as retrieving the sum count of '+' and '-' reads (and unknown strand reads in case these are found in the bam file). '*' is the default behaviour and can be used when the RNA-seq experiments strand information is not available.
getAlleleQuality

Value

getAlleleCounts returns a list of several data.frame objects, each storing the count data for one SNP.

Author(s)

Jesper R. Gadin, Lasse Folkersen

See Also

• The scanForHeterozygotes which is a function to find possible heterozygote sites in a GAlignmentsList object

Examples

#load example data
data(reads)
data(GRvariants)

#get counts at the three positions specified in GRvariants
alleleCount <- getAlleleCounts(BamList=reads,GRvariants,
strand="\'")

#if the reads had contained stranded data, these two calls would
#have given the correct input objects for getAlleleCounts
alleleCountPlus <- getAlleleCounts(BamList=reads,GRvariants,
strand="+")
alleleCountMinus <- getAlleleCounts(BamList=reads,GRvariants,
strand="-"")

getAlleleQuality 

snp quality data

Description

Given the positions of known SNPs, this function returns allele quality from a BamGRL object

Usage

getAlleleQuality(BamList, ...)

## S4 method for signature 'GAlignmentsList'
getAlleleQuality(BamList, GRvariants,
fastq.format = "illumina.1.8", return.class = "array", verbose = TRUE, ...

## S4 method for signature 'BamGRL'
getAlleleQuality(BamGRL, ...

## S4 method for signature 'BamGRL'
getAlleleQuality(BamGRL, GRvariants,
fastq.format = "illumina.1.8", return.class = "array", verbose = TRUE, ...
getAlleleQuality

Arguments

BamList A GAlignmentsList object or GRangesList object containing data imported from a bam file

... parameters to pass on

GRvariants A GRanges object that contains positions of SNPs to retrieve.

fastq.format default 'illumina.1.8'

return.class 'list' or 'array'

verbose Setting verbose=TRUE makes function more talkative

Details

This function is used to retrieve the allele quality strings from specified positions in a set of RNA-seq reads. The BamList argument will typically have been created using the impBamGAL function on bam-files. The GRvariants is either a GRanges with user-specified locations or else it is generated through scanning the same bam-files as in BamList for heterozygote locations (e.g. using scanForHeterozygotes). The GRvariants will currently only accept locations having width=1, corresponding to bi-allelic SNPs. The strand type information will be kept in the returned object. If the strand is marked as unknown "*", it will be forced to the "+" strand.

Quality information is extracted from the BamList object, and requires the presence of mcols(BamList)["qual"] to contain quality sequences.

Value

getAlleleQuality returns a list of several data.frame objects, each storing the count data for one SNP.

Author(s)

Jesper R. Gadin, Lasse Folkersen

Examples

#load example data
data(reads)
data(GRvariants)

#get counts at the three positions specified in GRvariants
alleleQualityArray <- getAlleleQuality(BamList=reads,GRvariants)

#place in ASEset object
alleleCountsArray <- getAlleleCounts(BamList=reads,GRvariants, strand="*", return.class="array")

a <- ASEsetFromArrays(GRvariants, countsUnknown = alleleCountsArray)
aquals(a) <- alleleQualityArray
getAreaFromGeneNames

Description

Given a character vector with genesymbols and an OrgDb object, this function returns a GRanges giving the coordinates of the genes.

Usage

getAreaFromGeneNames(genesymbols, ...)

## S4 method for signature 'character'
getAreaFromGeneNames(genesymbols, OrgDb, leftFlank = 0,
                 rightFlank = 0, na.rm = FALSE, verbose = TRUE)

Arguments

genesymbols A character vector that contains genesymbols of genes from which we wish to retrieve the coordinates

... arguments to pass on

OrgDb An OrgDb object containing gene annotation

leftFlank A integer specifying number of additional nucleotides before the genes

rightFlank A integer specifying number of additional nucleotides after the genes

na.rm A boolean removing genes that returned NA from the annotation

verbose Setting verbose=TRUE makes function more talkative

Details

This function is a convenience function that can be used to determine which genomic coordinates to specify to e.g. impBamGAL when retrieving reads.

The function cannot handle genes that do not exist in the annotation. To remove these please set the na.rm=TRUE.

Value

getAreaFromGeneNames returns a GRanges object with genomic coordinates around the specified genes

Author(s)

Jesper R. Gadin, Lasse Folkersen
**Examples**

```r
# load example data
data(ASEset)

# get counts at the three positions specified in GRvariants
library(org.Hs.eg.db)
searchArea <- getAreaFromGeneNames(c("PAX8", "TLR7"), org.Hs.eg.db)
```

**getDefaultMapBiasExpMean**

*Map Bias*

**Description**

an allele frequency array

**Usage**

```r
getDefaultMapBiasExpMean(alleleCountList, ...)
```

```r
getDefaultMapBiasExpMean3D(alleleCountList, ...)
```

```r
## S4 method for signature 'list'
getDefaultMapBiasExpMean(alleleCountList)
```

```r
## S4 method for signature 'ANY'
getDefaultMapBiasExpMean3D(alleleCountList)
```

**Arguments**

- `alleleCountList` A GRangesList object containing read information
- `...` parameters to pass on

**Details**

This function will assume there is no bias that comes from the mapping of reads, and therefore create a matrix with expected frequency of 0.5 for each allele.

**Value**

`getDefaultMapBiasExpMean` returns a matrix with a default expected mean of 0.5 for every element.

**Author(s)**

Jesper R. Gadin, Lasse Folkersen
Examples

```r
#load example data
data(ASEset)
#access SnpAfList
alleleCountList <- alleleCounts(ASEset)
#get default map bias exp mean
matExpMean <- getDefaultMapBiasExpMean(alleleCountList)
```

Description

Given a GRanges object of SNPs and a SNPlocs annotation, this function attempts to replace the names of the GRanges object entries with rs-IDs.

Usage

```r
getSnpIdFromLocation(GR, ...)  
## S4 method for signature 'GRanges'
getSnpIdFromLocation(GR, SNPloc, return.vector = FALSE, verbose = TRUE)
```

Arguments

- **GR**: A GRanges that contains positions of SNPs to look up
- **...**: arguments to pass on
- **SNPloc**: A SNPlocs object containing information on SNP locations (e.g. SNPlocs.Hsapiens.dbSNP.xxxxxxxx)
- **return.vector**: Setting return.vector=TRUE returns vector with rsIds
- **verbose**: Setting verbose=TRUE makes function more talkative

Details

This function is used to try to identify the rs-IDs of SNPs in a GRanges object.

Value

getSnpIdFromLocation returns the same GRanges object it was given with, but with updated with rs.id information.

Author(s)

Jesper R. Gadin, Lasse Folkersen
**GlobalAnalysis-class**

**Description**

Object that holds results from a global AI analysis including reference bias estimations and AI detection.

**Arguments**

- `x`: ASESet object or list of ASEsets
- `TxDb`: A transcriptDb object
- `...`: pass arguments to internal functions

**Details**

The GlobalAnalysis-class contains summaries and "pre-configured and pre-calculated lattice plots" needed to create an AI-report

**Value**

An object of class GlobalAnalysis containing all data to make report.

**Author(s)**

Jesper R. Gadin, Lasse Folkersen

**Examples**

```r
data(ASEset)
# a <- ASEset
#gba <- gba(a)

#report(gba)
#write.tables(gba)
#graphs(gba)
#as.list(gba)
```
GRvariants

GRvariants object

Description

This data is a GRanges object that contains the ranges for three example SNPs.

Author(s)

Jesper R. Gadin, Lasse Folkersen

See Also

- The reads which is another example object

Examples

```r
# load example data
data(GRvariants)
```

histplot

histogram plots

Description

Uses base graphics hist plot

Usage

```r
## S4 method for signature 'ASEset'
hist(x, strand = "+", type = "mean", log = 1, ...)
```

Arguments

- `x` : ReferenceBias object or ASEset object
- `strand` : \'+', '-' or '*' 
- `type` : 'mean' (only one option atm)
- `log` : an integer to log each value (integer 10 for log10)
- `...` : arguments to forward to internal boxplots function

Details

The histogram will show the density over frequencies for each sample

Author(s)

Jesper R. Gadin, Lasse Folkersen
Examples

```r
##load example data
#data(ASEset)
a <- ASEset
#hist(a)
```

### implodeList.old

**implode list of arguments into environment**

**Description**

apply on list of variables to be put in the local environment

**Usage**

```r
implodeList.old(x)
```

**Arguments**

- **x**  
  list of variables

**Details**

help the propagation of e.g. graphical paramters

**Author(s)**

Jesper R. Gadin

**Examples**

```r
lst <- list(hungry='yes', thirsty='no')
implodeList.old(lst)
#the check ls()
ls()
```
import-bam

Import Bam

Description
Imports a specified genomic region from a bam file using a GRanges object as search area.

Usage
impBamGAL(UserDir, ...)

## S4 method for signature 'character'
impBamGAL(UserDir, searchArea, files = NULL, XStag = FALSE, verbose = TRUE, ...)

Arguments
- **UserDir**: The relative or full path of folder containing bam files.
- **searchArea**: A GenomicRanges object that contains the regions of interest.
- **files**: use character vector to specify one or more files to import. The default imports all bam files from the directory.
- **XStag**: Setting XStag=TRUE stores the strand specific information in the mcols slot 'XS'.
- **verbose**: makes the function more talkative.

Details
If the sequence data is strand-specific you may want to set XStag=TRUE. The strand specific information has then to be stored in the meta columns with column name ‘XS’. If the aligner did not set the XS-tag and the data is strand-specific it is still be possible to infer the strand from the bit flags after importing the reads to R. Depending on the strand-specific protocol different combinations of the flags will have to be used. For illumina fr-secondstrand, 83 and 163 are minus strand reads and 99 and 147 are plus strand reads.

Author(s)
Jesper R. Gadin, Lasse Folkersen

Examples

# Declare searchArea
searchArea <- GRanges(seqnames=c('17'), ranges=IRanges(79478301,79478361))

# Relative or full path
pathToFiles <- system.file('extdata/ERP000101_subset', package='AllelicImbalance')

# all files in directory
reads <- impBamGAL(pathToFiles, searchArea, verbose=FALSE)
# specified files in directory
reads <- impBamGAL(pathToFiles, searchArea,
import-bam-2

Description
Imports bla bal bal a specified genomic region from a bam file using a GenomicRanges object as search area.

Usage
impBamGRL.old(UserDir, searchArea, verbose = TRUE)

Arguments
UserDir The relative or full path of folder containing bam files.
searchArea A GenomicRanges object that contains the regions of interest
verbose Setting verbose=TRUE gives details of procedure during function run.

Details
These functions are right on tahea wrappers to import bam files into R and store them into either GRanges, GAlignments or GappedAlignmentpairs objects.

It is recommended to use the impBamGAL() which takes information of gaps into account. It is also possible to use the other variants as well, but then pre-filtering becomes important keeps to understand because gapped, intron-spanning reads will cause problems. This is because the GRanges objects can not handle if gaps are present and will then give a wrong result when calculating the allele (SNP) count table.

Value
impBamGRL returns a GRangesList object containing the RNA-seq reads in the region defined by the searchArea argument. impBamGAL returns a list with GAlignments objects containing the RNA-seq reads in the region defined by the searchArea argument. funImpBamGAPL returns a list with GappedAlignmentPairs object containing the RNA-seq reads in the region defined by the searchArea argument.

Author(s)
Jesper R. Gadin, Lasse Folkerensen

Examples
#Declare searchArea
searchArea <- GRanges(seqnames=c('17'), ranges=IRanges(79478301,79478361))

#Relative or full path
pathToFiles <- system.file('extdata/ERP000101_subset', package='AllelicImbalance')
**import-bcf**  

**Import Bcf Selection**

**Description**
Imports a selection of a bcf file or files specified by a GenomicRanges object as search area.

**Usage**

```r
impBcfGRL(UserDir, ...)  
## S4 method for signature 'character'
impBcfGRL(UserDir, searchArea = NULL, verbose = TRUE, ...

impBcfGR(UserDir, ...)
## S4 method for signature 'character'
impBcfGR(UserDir, searchArea = NULL, verbose = TRUE, ...
```

**Arguments**

- `UserDir` The relative or full path of folder containing bam files.
- `...` parameters to pass on
- `searchArea` A GenomicRanges object that contains the regions of interest
- `verbose` Setting verbose=TRUE gives details of the procedure during function run.

**Details**
A wrapper to import bcf files into R in the form of GenomicRanges objects.

**Value**

BcfImpGRL returns a GRangesList object. BcfImpGR returns one GRanges object of all unique entries from one or more bcf files.

**Note**
Make sure there is a complementary index file *.bcf.bci for each bcf file in UserDir. If there is not, then the functions impBcfGRL and impBcfGR will try to create them.

**Author(s)**
Jesper R. Gadin, Lasse Folkersen

**See Also**
- The impBamGRL for importing bam files
- The `getAlleleCounts` for how to get allele(SNP) counts
- The `scanForHeterozygotes` for how to find possible heterozygote positions
# Examples

```r
# Declare searchArea
searchArea <- GRanges(seqnames=c('17'), ranges=IRanges(79478301,79478361))

# Relative or full path
pathToFiles <- system.file('extdata/ERP000101_subset', package='AllelicImbalance')

# Import
reads <- impBcfGRL(pathToFiles, searchArea, verbose=FALSE)
```

---

**inferAlleles**

**inference of SNPs of ASEset**

### Description

inference of SNPs

### Usage

`inferAlleles(x, strand = "\*", return.type = "bi", threshold.frequency = 0, threshold.count.sample = 1, inferOver = "eachSample", allow.NA = FALSE)`

### Arguments

- `x` : ASEset
- `strand` : strand to infer from
- `return.type` : 'uni' 'bi' 'tri' 'quad' 'all'
- `threshold.frequency` : least fraction to classify (see details)
- `threshold.count.sample` : least amount of counts to try to infer allele
- `inferOver` : 'eachSample' or 'allSamples'
- `allow.NA` : treat NA as zero when TRUE

### Details

`threshold.frequency` is the least fraction needed to classify as bi tri or quad allelic SNPs. If 'all' then all of bi tri and quad allelic SNPs will use the same threshold. Everything under the treshold will be regarded as noise. 'all' will return a matrix with snps as rows and uni bi tri and quad will be columns. For this function Anything that will return TRUE for tri-allelic will also return TRUE for uni and bi-allelic for the same SNP an Sample.

### Author(s)

Jesper R. Gadin
**inferAltAllele**

### Examples

```r
data(ASEset)
i <- inferAlleles(ASEset)
```

---

**Description**

inference of the alternate allele based on count data

### Arguments

- `x` matrix see examples
- `return.class` class of returned object
- `allele.source` 'arank'
- `verbose` make function more talkative
- `...` arguments to forward to internal functions

### Details

The inference essentially ranks all alleles and the most expressed allele not declared as reference will be inferred as the alternative allele. At the moment only inference of bi-allelic alternative alleles are available.

### Author(s)

Jesper R. Gadin, Lasse Folkersen

### Examples

```r
# load data
data(ASEset)

alt <- inferAltAllele(ASEset)
```
inferGenotypes

inferGenotypes is an R function for inferring genotypes from ASEset count data.

### Description

Inference of genotypes

### Usage

```r
inferGenotypes(x, strand = "*", return.class = "matrix", 
    return.allele.allowed = "bi", threshold.frequency = 0, 
    threshold.count.sample = 1)
```

### Arguments

- **x**: ASEset
- **strand**: strand to infer from
- **return.class**: 'matrix' or 'vector'
- **return.allele.allowed**: vector with 'bi' 'tri' or 'quad'. 'uni' always gets returned
- **threshold.frequency**: least fraction to classify (see details)
- **threshold.count.sample**: least amount of counts to try to infer allele

### Details

Often necessary information to link AI to SNPs outside coding region

### Author(s)

Jesper R. Gadin

### Examples

```r
data(ASEset)
g <- inferGenotypes(ASEset)
```
initialize-ASEset

Description

Functions to construct ASEset objects

Usage

ASEsetFromCountList(rowRanges, countListUnknown = NULL, countListPlus = NULL, countListMinus = NULL, colData = NULL, mapBiasExpMean = NULL, phase = NULL, aquals = NULL, verbose = FALSE, ...)

ASEsetFromArray(rowRanges, countsUnknown = NULL, countsPlus = NULL, countsMinus = NULL, colData = NULL, mapBiasExpMean = NULL, phase = NULL, genotype = NULL, aquals = NULL, verbose = FALSE, ...)

Arguments

rowRanges A GenomicRanges object that contains the variants of interest
countListUnknown A list where each entry is a matrix with allele counts as columns and sample counts as rows
countListPlus A list where each entry is a matrix with allele counts as columns and sample counts as rows
countListMinus A list where each entry is a matrix with allele counts as columns and sample counts as rows
colData A DataFrame object containing sample specific data
mapBiasExpMean A 3-D array where the SNPs are in the 1st dimension, samples in the 2nd dimension and variants in the 3rd dimension.
phase A matrix or an array containing phase information.
aquals A 4-D array containing the countinformation, see details
verbose Makes function more talkative
... arguments passed on to SummarizedExperiment constructor
countsUnknown An array containing the countinformation
countsPlus An array containing the countinformation
countsMinus An array containing the countinformation
genotype matrix

details

The resulting ASEset object is based on the RangedSummarizedExperiment class, and will therefore inherit the same accessors and ranges operations.

If both countListPlus and countListMinus are given they will be used to calculate countListUnknown, which is the sum of the plus and minus strands.
countListPlus, countListMinus and countListUnknown are i.e. the outputs from the getAlleleCounts function. aquals is new for the devel branch and will be changed slightly before the release to include better granularity.

Value

ASEsetFromCountList returns an ASEset object.

Note

ASEsetFromCountList requires the same input data as a RangedSummarizedExperiment, but with minimum one assay for the allele counts.

Author(s)

Jesper R. Gadin, Lasse Folkersen

See Also

• RangedSummarizedExperiment objects.

Examples

# make example alleleCountListPlus
set.seed(42)
countListPlus <- list()
snps <- c('snp1','snp2','snp3','snp4','snp5')
for(snp in snps){
count <- matrix(rep(0,16),ncol=4,dimnames=list(
c('sample1','sample2','sample3','sample4'),
c('A','T','G','C')))
# insert random counts in two of the alleles
for(allele in sample(c('A','T','G','C'),2)){
count[,allele] <- as.integer(rnorm(4,mean=50,sd=10))
}
countListPlus[[snp]] <- count
}

# make example alleleCountListMinus
countListMinus <- list()
snps <- c('snp1','snp2','snp3','snp4','snp5')
for(snp in snps){
count <- matrix(rep(0,16),ncol=4,dimnames=list(
c('sample1','sample2','sample3','sample4'),
c('A','T','G','C')))
# insert random counts in two of the alleles
for(allele in sample(c('A','T','G','C'),2)){
count[,allele] <- as.integer(rnorm(4,mean=50,sd=10))
}
countListMinus[[snp]] <- count
}

# make example rowRanges
initialize-DetectedAI

```
rowRanges <- GRanges(
  seqnames = Rle(c('chr1', 'chr2', 'chr3', 'chr1', 'chr1'),
  ranges = IRanges(1:5, width = 1, names = head(letters,5)),
  snp = paste('snp',1:5,sep='')
)

# make example colData
colData <- DataFrame(Treatment=c('ChIP', 'Input','Input','ChIP'),
  row.names=c('ind1','ind2','ind3','ind4'))

# make ASEset
a <- ASEsetFromCountList(rowRanges, countListPlus=countListPlus,
  countListMinus=countListMinus, colData=colData)
```

---

initialize-DetectedAI  
*Initialize DetectedAI*

**Description**

Functions to construct DetectedAI objects

**Usage**

```
DetectedAIFromArray(x = "ASEset", strand = "*",
  reference.frequency = NULL, threshold.frequency = NULL,
  threshold.count.sample = NULL, threshold.delta.frequency = NULL,
  threshold.pvalue = NULL, threshold.frequency.names = NULL,
  threshold.count.sample.names = NULL,
  threshold.delta.frequency.names = NULL, threshold.pvalue.names = NULL,
  ...
)
```

**Arguments**

- `x`  
  ASEset
- `strand`  
  set strand to detectAI over "+","-","*"
- `reference.frequency`  
  frequencies of reference alleles based allele counts
- `threshold.frequency`  
  logical array for frequency thresholds
- `threshold.count.sample`  
  logical array for per sample allele count thresholds
- `threshold.delta.frequency`  
  logical array for delta frequency thresholds.
- `threshold.pvalue`  
  logical array for pvalue thresholds (max 1, min 0)
- `threshold.frequency.names`  
  character vector
- `threshold.count.sample.names`  
  character vector
- `threshold.delta.frequency.names`  
  character vector
- `threshold.pvalue.names`  
  character vector
initialize-GlobalAnalysis

threshold.pvalue.names
character vector
...
internal arguments

Details
produces a class container for reference bias calculations

Author(s)
Jesper R. Gadin, Lasse Folkersen

Examples

data(ASEset)
a <- ASEset
dai <- detectAI(a)

initialize-GlobalAnalysis

Initialize GlobalAnalysis

Description
Functions to construct GlobalAnalysis objects

Usage
GAnalysis(x = "ASEset", ...)

Arguments
x
ASEset
...
internal arguments

Details
produces a class container for a global analysis

Author(s)
Jesper R. Gadin, Lasse Folkersen

Examples

data(ASEset)
a <- ASEset
# gba <- gba(a)
**Initialize RiskVariant**

*Description*

Functions to construct RiskVariant objects

*Usage*

```r
RiskVariantFromGRangesAndPhaseArray(x, phase, ...)
```

*Arguments*

- `x` : GRanges object for the SNPs
- `phase` : array with phaseinfo
- `...` : internal arguments

*Details*

produces a class container for reference bias calculations

*Author(s)*

Jesper R. Gadin, Lasse Folkersen

*Examples*

```r
data(ASEset)
#p <- getPhaseFromSomewhere
#rv <- RiskVariantFromGRangesAndPhaseArray(x=GRvariants, phase=p)
```

---

**legendBarplot**

*Description*

adds a very customizable legend function for AllelicImbalance barplots.

*Usage*

```r
legendBarplot(lowerLeftCorner, size, rownames, colnames, boxsize = 1,
boxspace = 1, fgCol, bgCol, ylegendPos = 1, xlegendPos = 0.96,
cex = 1)
```
**Arguments**

- `lowerLeftCorner`:
  position of the plot to add legend to (default c(0,0))
- `size`:
  scale the plot, default is 1
- `rownames`:
  rownames in legend
- `colnames`:
  colnames in legend
- `boxsize`:
  size of each box fill
- `boxspace`:
  space inbetween the box fill
- `fgCol`:
  color for allele1
- `bgCol`:
  color for allele2
- `ylegendPos`:
  placement of the legend within the plot for y
- `xlegendPos`:
  placement of the legend within the plot for x
- `cex`:
  size of legend text

**Details**

the function is preferably called from within the AllelicImbalance barplot method.

**Author(s)**

Jesper R. Gadin

**Examples**

```r
# code placeholders
#< create a barplot with legend >
#< add legend >
```

---

**Description**

Object that holds results from AI detection.

**Usage**

```r
pvalue(x, ...)
```

## S4 method for signature 'LinkVariantAlmlof'

```r
pvalue(x)
```

**Arguments**

- `x`:
  LinkVariantAlmlof object
- `...`:
  pass arguments to internal functions
The LinkVariantAlmlof-class contains

Author(s)
Jesper R. Gadin, Lasse Folkersen

Examples

#some code

---

**Description**
plot an object of type LinkVariantAlmlof

**Usage**
plot(x, y, ...)  
## S4 method for signature 'LinkVariantAlmlof,ANY'
plot(x, y, ...)

**Arguments**
x LinkVariantAlmlof object
y not used
... pass on arguments to internal methods

**Author(s)**
Jesper R. Gadin, Lasse Folkersen

**Examples**

data(ASEset)
a <- ASEset
# Add phase
set.seed(1)
p1 <- matrix(sample(c(1,0),replace=TRUE, size=nrow(a)*ncol(a)),nrow=nrow(a), ncol(a))
p2 <- matrix(sample(c(1,0),replace=TRUE, size=nrow(a)*ncol(a)),nrow=nrow(a), ncol(a))
p <- matrix(paste(p1,sample(c("\",",","/"), size=nrow(a)*ncol(a), replace=TRUE), p2, sep=""), nrow=nrow(a), ncol(a))
phase(a) <- p
# add alternative allele information
mcols(a)[["alt"]]["alt"] <- inferAltAllele(a)

# init risk variants
p.ar <- phaseMatrix2Array(p)
rv <- RiskVariantFromGRangesAndPhaseArray(x=GRvariants, phase=p.ar)

# colnames has to be same and same order in ASEset and RiskVariant
colnames(a) <- colnames(rv)

# in this example each and every snp in the ASEset defines a region
r1 <- granges(a)

# in this example two overlapping subsets of snps in the ASEset defines the region
r2 <- split(granges(a)[c(1,2,2,3),c(1,1,2,2)])

# link variant almlof (lva)
lv1 <- lva(a, rv, r1)
lv2 <- lva(a, rv, r2)
plot(lv2[1])

---

### lva

Description

make an almlof regression for arrays

Usage

```r
lva(x, ...)
```

### S4 method for signature 'ASEset'

```r
lva(x, rv, region, settings = list(),
    return.class = "LinkVariantAlmlof", verbose = FALSE, ...)
```

Arguments

- `x` ASESet object with phase and 'ref'/alt' allele information
- `...` arguments to forward to internal functions
- `rv` RiskVariant object with phase and 'ref'/alt' allele information
- `region` RiskVariant object with phase and alternative allele information
- `settings` RiskVariant object with phase and alternative allele information
- `return.class` 'LinkVariantAlmlof' (more options in future)
- `verbose` logical, if set TRUE, then function will be more talkative

Details

internal method that takes one array with results from regionSummary and one matrix with group information for each risk SNP (based on phase)
Author(s)

Jesper R. Gadin, Lasse Folkesen

Examples

```r
# Add phase
p1 <- matrix(sample(c(1,0),replace=TRUE, size=nrow(a)*ncol(a)),nrow=nrow(a), ncol(a))
p2 <- matrix(sample(c(1,0),replace=TRUE, size=nrow(a)*ncol(a)),nrow=nrow(a), ncol(a))
p <- matrix(paste(p1,sample(c("\|","\|","/"), size=nrow(a)*ncol(a), replace=TRUE), p2, sep=""),
nrow=nrow(a), ncol(a))
phase(a) <- p

#add alternative allele information
mcols(a)[["alt"]] <- inferAltAllele(a)

#init risk variants
p.ar <- phaseMatrix2Array(p)
rv <- RiskVariantFromGRangesAndPhaseArray(x=GRvariants, phase=p.ar)

#colnames has to be samea and same order in ASEset and RiskVariant
colnames(a) <- colnames(rv)

# in this example each and every snp in the ASEset defines a region
r1 <- granges(a)

#use GRangesList to merge and use regions defined by each element of the
#GRangesList
r1b <- GRangesList(r1)
r1c <- GRangesList(r1, r1)

# in this example two overlapping subsets of snps in the ASEset defines the region
r2 <- split(granges(a)[c(1,2,2,3)],c(1,1,2,2))

# link variant almlof (lva)
lva(a, rv, r1)
lva(a, rv, r1b)
lva(a, rv, r1c)
lva(a, rv, r2)
```

Description

make an almlof regression for arrays (internal function)
Usage

lva.internal(x, ...)

## S4 method for signature 'array'
lva.internal(x, grp, element = 3, ...)

Arguments

x      regionSummary array phased for maternal allele
...
    arguments to forward to internal functions
grp    group 1-3 (1 for 0:0, 2 for 1:0 or 0:1, and 3 for 1:1)
element which column in x contains the values to use with lm.

Details

internal method that takes one array with results from regionSummary and one matrix with group information for each risk SNP (based on phase). Input and output objects can change format slightly in future.

Author(s)

Jesper R. Gadin, Lasse Folkersen

Examples

data(ASEset)
a <- ASEset
# Add phase
set.seed(1)
p1 <- matrix(sample(c(1,0),replace=TRUE, size=nrow(a)*ncol(a)),nrow=nrow(a), ncol(a))
p2 <- matrix(sample(c(1,0),replace=TRUE, size=nrow(a)*ncol(a)),nrow=nrow(a), ncol(a))
p <- matrix(paste(p1,sample(c("\","","/"), size=nrow(a)*ncol(a), replace=TRUE), p2, sep=""),
nrow=nrow(a), ncol(a))
phase(a) <- p

#add alternative allele information
cols(a)[["alt"]]<- inferAltAllele(a)

# in this example two overlapping subsets of snps in the ASEset defines the region
region <- split(granges(a)[c(1,2,2,3), c(1,1,2,2)])
rs <- regionSummary(a, region, return.class="array", return.meta=FALSE)

# use (change to generated riskSNP phase later)
phs <- array(c(phase(a,return.class="array")[1,,c(1, 2)],
    phase(a,return.class="array")[2,,c(1, 2)], dim=c(20,2,2))
grp <- matrix(2, nrow=dim(phs)[1], ncol=dim(phs)[2])
grp[(phs[,,1] == 0) & (phs[,,2] == 0)] <- 1
grp[(phs[,,1] == 1) & (phs[,,2] == 1)] <- 3
#only use mean.fr at the moment, which is col 3
lva.internal(asasys(rs)[["rs1"]], grp, 3)
makeMaskedFasta  makes masked fasta reference

Description
Replaces all selected positions in a fasta file with the character N

Usage
makeMaskedFasta(fastaIn, ...)

## S4 method for signature 'character'
makeMaskedFasta(fastaIn, fastaOut, posToReplace,
    splitOnSeqlevels = TRUE, verbose = TRUE)

Arguments
- fastaIn: character string of the path for the fasta file to be used
- ...: arguments to pass on
- fastaOut: character string of the path for the masked fasta file (no extension)
- posToReplace: GRanges object with the genomic ranges to replace
- splitOnSeqlevels: write on file for each seqlevel to save memory
- verbose: makes function more talkative

Author(s)
Jesper R. Gadin

Examples

data(ASEset.sim)
gr <- rowRanges(ASEset.sim)
fastaIn <- system.file('extdata/hg19.chr17.subset.fa', package='AllelicImbalance')
makeMaskedFasta(fastaIn=fastaIn, fastaOut="fastaOut", posToReplace=gr)

mapBiasRef  mapBias for reference allele

Description
Create a matrix of bias for the reference allele
Usage

mapBiasRef(x, ...)

### S4 method for signature 'ASEset'
mapBiasRef(x)

Arguments

- `x` ASEset object
- `...` internal arguments

Details

select the expected frequency for the reference allele

Author(s)

Jesper R. Gadin, Lasse Folkersen

Examples

```r
# load example data
data(ASEset)
a <- ASEset
mat <- mapBiasRef(a)
```

---

Description

filter on minCountFilt snps

Usage

minCountFilt(x, ...)

### S4 method for signature 'ASEset'
minCountFilt(x, strand = "*", threshold.counts = 1,
         sum = "all", replace.with = "zero", return.class = "ASEset")

Arguments

- `x` ASEset object
- `...` internal param
- `strand` strand to infer from
- `threshold.counts` cutoff for read counts (see details)
minFreqFilt

sum: 'each' or 'all'
replace.with: only option 'zero'
return.class: 'ASEset', 'array' or 'matrix'

Details
Description info here

Author(s)
Jesper R. Gadin, Lasse Folkersen

Examples

# load example data
data(ASEset)
a <- ASEset

minCountFilt(a)

Description
filter on minFreqFilt snps

Usage

minFreqFilt(x, ...)

### S4 method for signature 'ASEset'
minFreqFilt(x, strand = "*", threshold.frequency = 0.1,
replace.with = "zero", return.class = "ASEset", sum = "all")

Arguments

x: ASEset object
...
strand: strand to infer from
threshold.frequency: least fraction to classify (see details)
replace.with: only option 'zero'
return.class: 'ASEset', 'array' or 'matrix'
sum: 'each' or 'all'

Details
Description info here
multiAllelicFilt

Author(s)
Jesper R. Gadin, Lasse Folkersen

Examples

```r
#load example data
data(ASEset)
a <- ASEset
minFreqFilt(a)
```

Description
filter on multiallelic snps

Usage

```r
multiAllelicFilt(x, ...)

## S4 method for signature 'ASEset'
multiAllelicFilt(x, strand = "*",
    threshold.count.sample = 10, threshold.frequency = 0.1,
    filterOver = "eachSample")
```

Arguments

- `x` ASEset object
- `...` internal param
- `strand` strand to infer from
- `threshold.count.sample` least amount of counts to try to infer allele
- `threshold.frequency` least fraction to classify (see details)
- `filterOver` 'eachSample' or 'allSamples'

Details
based on the allele counts for all four variants A, T, G and C and returns true if there is counts enough suggesting a third or more alleles. The sensitivity can be specified using 'threshold.count.sample' and 'threshold.frequency'.

Author(s)
Jesper R. Gadin, Lasse Folkersen
Examples

# load example data
data(ASEset)
a <- ASEset
multiAllelicFilt(a)

Description

Convert the phase from the internally stored phase, ref and alt information

Usage

phase2genotype(x, ...)

## S4 method for signature 'array'
phase2genotype(x, ref, alt, return.class = "matrix", ...)

Arguments

x array see examples
...
ref reference allele vector
alt alternative allele vector
return.class 'matrix' or 'array'

Details

To not introduce redundant information in the ASEset object, the genotype matrix is accessed from the phase matrix, which together with ref and alt allele information contains the same information (not taken into account three-allelic or more SNPs).

The genotype matrix retrieved from an ASEset object can differ from the genotype matrix stored in the object if reference and alternative alleles were not used or has changed since the phase genotype matrix was stored. Basically, it is preferable to provide reference and alternative information when storing the genotype matrix.

If possible, it is better to not use a genotype matrix, but instead relying completely on storing a phase matrix (or array) together with reference and alternative allele information.

Author(s)

Jesper R. Gadin, Lasse Folkersen
Examples

# load example data
data(ASEset)
data(genomatrix)
p <- genotype2phase(genomatrix, ref(ASEset), return.class="array")
ref <- ref(ASEset)
alt <- inferAltAllele(ASEset)

gt <- phase2genotype(p, ref, alt, return.class="matrix")

phaseArray2phaseMatrix

Description

used to convert the phase from the visually friendly matrix to array.

Usage

phaseArray2phaseMatrix(x, ...)

# S4 method for signature 'array'
phaseArray2phaseMatrix(x, ...)

Arguments

x array see examples
...

arguments to forward to internal functions

Details

A more effectice way of store the phase data in the ASEset object

Author(s)

Jesper R. Gadin, Lasse Folkersen

Examples

# load data
data(ASEset)
a <- ASEset

# example phase matrix
p1 <- matrix(sample(c(1,0),replace=TRUE, size=nrow(a)*ncol(a)),nrow=nrow(a), ncol(a))
p2 <- matrix(sample(c(1,0),replace=TRUE, size=nrow(a)*ncol(a)),nrow=nrow(a), ncol(a))
p <- matrix(paste(p1,sample(c("|","|","/"), size=nrow(a)*ncol(a), replace=TRUE), p2, sep=""),
nrow=nrow(a), ncol(a))
phaseMatrix2Array

ar <- phaseMatrix2Array(p)

#Convert back
mat <- phaseArray2phaseMatrix(ar)

---

phaseMatrix2Array  phaseMatrix2Array

Description

used to convert the phase from the visually friendly matrix to array.

Usage

phaseMatrix2Array(x, ...)

## S4 method for signature 'matrix'
phaseMatrix2Array(x, dimnames = NULL, ...)

Arguments

x  matrix see examples
...
arguments to forward to internal functions
dimnames  list with dimnames

Details

A more effective way of store the phase data in the ASEset object

Author(s)

Jesper R. Gadin, Lasse Folkersen

Examples

#load data
data(ASEset)
a <- ASEset

#example phase matrix
p1 <- matrix(sample(c(1,0),replace=TRUE, size=nrow(a)*ncol(a)),nrow=nrow(a), ncol(a))
p2 <- matrix(sample(c(1,0),replace=TRUE, size=nrow(a)*ncol(a)),nrow=nrow(a), ncol(a))
p <- matrix(paste(p1,sample(c("\","\","/"), size=nrow(a)*ncol(a), replace=TRUE), p2, sep=""),
nrow=nrow(a), ncol(a))
ar <- phaseMatrix2Array(p)
randomRef  

Random ref allele from genotype

Description

Create a vector of random reference alleles

Usage

```r
randomRef(x, ...)  
## S4 method for signature 'ASEset'
randomRef(x, source = "alleleCounts", ...)
```

Arguments

- `x`  
  ASEset object
- `...`  
  internal arguments
- `source`  
  'alleleCounts'

Details

Randomly shuffles which of the two alleles for each genotype that is indicated as reference allele, based on either allele count information or previous ref and alt alleles.

When the source is 'alleleCounts', the two most expressed alleles are taken as reference and alternative allele.

Author(s)

Jesper R. Gadin, Lasse Folkersen

Examples

```r
# load example data
data(ASEset.sim)
a <- ASEset.sim

ref(a) <- randomRef(a, source = 'alleleCounts')
```
reads

Description

This data set corresponds to the BAM-file data import illustrated in the vignette. The data set consists of a chromosome 17 region from 20 RNA-seq experiments of HapMap samples.

Author(s)

Jesper R. Gadin, Lasse Folksersen

References


See Also

• The GRvariants which is another example object

Examples

##load example data (Not Run)
#data(reads)

refAllele

Description

Extract the allele based on SNP location from the reference fasta file

Usage

refAllele(x, fasta)

Arguments

x ASEset object

fasta path to fasta file, index should be located in the same folder

Details

The alleles will be placed in the rowRanges() meta column 'ref'

Author(s)

Jesper R. Gadin, Lasse Folksersen
Examples

```r
# load example data
data(ASEset.sim)

fasta <- system.file('extdata/hg19.chr17.subset.fa', package='AllelicImbalance')
a <- refAllele(ASEset.sim,fasta=Fasta)
```

regionSummary

Description

Gives a summary of AI-consistency for a transcript

Usage

```r
regionSummary(x, ...)
```

## S4 method for signature 'ASEset'
regionSummary(x, region, strand = "*",
return.class = "RegionSummary", ...)

Arguments

- `x` : ASEset object
- `...` : arguments to forward to internal functions
- `region` : to summarize over, the object can be a GRanges, GRangesList
- `strand` : can be "+", "," or "*"
- `return.class` : "array" or "list".

Details

From a given set of e.g. transcripts exon ranges the function will return a summary for the sum of all exons. Phase information, reference and alternative allele is required.

A limitation comes to the strand-specificness. At the moment it is not possible to call over more than one strand type using the strands in region. This will be improved before going to release.

to calculate the direction and binomial p-values of AI the mapbias stored in the ASEset is used. see `?mapBias`.

Author(s)

Jesper R. Gadin, Lasse Folkersen
Examples

```r
data(ASEset)
a <- ASEset
# Add phase
set.seed(1)
p1 <- matrix(sample(c(1,0),replace=TRUE, size=nrow(a)*ncol(a)),nrow=nrow(a), ncol(a))
p2 <- matrix(sample(c(1,0),replace=TRUE, size=nrow(a)*ncol(a)),nrow=nrow(a), ncol(a))
p <- matrix(paste(p1,sample(c("|","|","/"), size=nrow(a)*ncol(a), replace=TRUE), p2, sep=""),
nrow=nrow(a), ncol(a))

phase(a) <- p

#add alternative allele information
mcols(a)[["alt"]]<- inferAltAllele(a)

# in this example each and all snps in the ASEset defines the region
region <- granges(a)
t <- regionSummary(a, region)

# in this example two overlapping subsets of snps in the ASEset defines the region
region <- split(granges(a)[c(1,2,2,3),c(1,1,2,2)])
t <- regionSummary(a, region)
```

RegionSummary-class  RegionSummary class

Description

Object that holds results from the regionSummary method

Usage

```r
sumnames(x, ...)

## S4 method for signature 'RegionSummary'
sumnames(x)

basic(x, ...)

## S4 method for signature 'RegionSummary'
basic(x)
```

Arguments

- `x` RegionSummary object
- `...` pass arguments to internal functions

Details

The RegionSummary-class objects contains summaries for specified regions
Author(s)
Jesper R. Gadin, Lasse Folkersen

Examples

```r
#some code
```

RiskVariant-class

Description
Object that holds results from AI detection.

Usage

```r
## S4 method for signature 'RiskVariant'
ref(x)

## S4 replacement method for signature 'RiskVariant,ANY'
ref(x) <- value

## S4 method for signature 'RiskVariant'
alt(x)

## S4 replacement method for signature 'RiskVariant,ANY'
alt(x) <- value

## S4 method for signature 'RiskVariant'
phase(x, return.class = "matrix")

## S4 replacement method for signature 'RiskVariant'
phase(x) <- value
```

Arguments

- `x` RiskVariant object or list of RiskVariants
- `value` argument used for replacement
- `return.class` type of class returned eg. "list or ""array".
- `...` pass arguments to internal functions

Details
The RiskVariant-class contains

Author(s)
Jesper R. Gadin, Lasse Folkersen
Examples

#some code

---

scanForHeterozygotes.old

**scanForHeterozygotes-old**

---

**Description**

Identifies the positions of SNPs found in BamGR reads.

**Usage**

```r
scanForHeterozygotes.old(BamList, minimumReadsAtPos = 20,
maximumMajorAlleleFrequency = 0.9, minimumBiAllelicFrequency = 0.9,
maxReads = 15000, verbose = TRUE)
```

**Arguments**

- `BamList` A `GAlignmentsList` object
- `minimumReadsAtPos` minimum number of reads required to call a SNP at a given position
- `maximumMajorAlleleFrequency` maximum frequency allowed for the most common allele. Setting this parameter lower will minimise the SNP calls resulting from technical read errors, at the cost of missing loci with potential strong ASE
- `minimumBiAllelicFrequency` minimum frequency allowed for the first and second most common allele. Setting a Lower value for this parameter will minimise the identification of loci with three or more alleles in one sample. This is useful if sequencing errors are suspected to be common.
- `maxReads` max number of reads of one list-element allowed
- `verbose` logical indicating if process information should be displayed

**Details**

This function scans all reads stored in a `GAlignmentsList` for possible heterozygote positions. The user can balance the sensitivity of the search by modifying the `minimumReadsAtPos`, `maximumMajorAlleleFrequency` and `minimumBiAllelicFrequency` arguments.

**Value**

`scanForHeterozygotes.old` returns a `GRanges` object with the SNPs for the `BamList` object that was used as input.

**Author(s)**

Jesper R. Gadin, Lasse Folkersen
See Also

- The `getAlleleCounts` which is a function that count the number of reads overlapping a site.

Examples

```r
data(reads)
s <- scanForHeterozygotes.old(reads, verbose=FALSE)
```
Index

*Topic **ASEDAAnnotationTrack**  
ASESet-gviztrack, 17

*Topic **ASESetFromCountList**  
initialize-ASESet, 53

*Topic **ASESet**  
ASESet-class, 9  
ASESetFromBam, 22  
DetectedAI-class, 31  
GlobalAnalysis-class, 44  
initialize-ASESet, 53

*Topic **CDS**  
annotation-wrappers, 4

*Topic **CoverageDataTrack**  
ASESet-gviztrack, 17

*Topic **LinkVariantAlmlof**  
LinkVariantAlmlof-class, 58

*Topic **RegionSummary**  
RegionSummary-class, 73

*Topic **RiskVariant**  
RiskVariant-class, 74

*Topic **SNP**  
ASESet-scanForHeterozygotes, 20  
getAlleleCounts, 38  
getSnplDFromLocation, 43  
scanForHeterozygotes.old, 75

*Topic **allelecount**  
countAllelesFromBam, 26

*Topic **allele**  
getAlleleQuality, 39

*Topic **annotation**  
annotation-wrappers, 4  
annotationBarplot, 6

*Topic **bam**  
import-bam, 47  
import-bam-2, 48

*Topic **barplot**  
annotationBarplot, 6  
ASESet-barplot, 7  
barplot-lattice-support, 23  
legendBarplot, 57

*Topic **bcf**  
import-bcf, 49

*Topic **bias**  
getDefaultMapBiasExpMean, 42  
initialize-DetectedAI, 55  
initialize-RiskVariant, 57

*Topic **binomial**  
binom.test, 24

*Topic **chi-square**  
chisq.test, 25

*Topic **class**  
ASESet-class, 9  
DetectedAI-class, 31  
GlobalAnalysis-class, 44  
LinkVariantAlmlof-class, 58  
RegionSummary-class, 73  
RiskVariant-class, 74

*Topic **counting**  
countAllelesFromBam, 26

*Topic **count**  
getAlleleCounts, 38

*Topic **coverage**  
coverageMatrixListFromGAL, 27

*Topic **data**  
ASESet.old, 21  
ASESet.sim, 22  
genomatrix, 37  
GRvariants, 45  
reads, 71

*Topic **detection**  
detectAI, 29

*Topic **example**  
ASESet.old, 21  
ASESet.sim, 22  
GRvariants, 45  
reads, 71

*Topic **exons**  
annotation-wrappers, 4

*Topic **fasta**  
makeMaskedFasta, 63

*Topic **filter**  
ASESet-filters, 14  
minCountFilt, 64  
minFreqFilt, 65  
multiAllelicFilt, 66

*Topic **gbarplot**
frequency, ASEset-method (ASEset-class), 9
frequency_vs_threshold_variable_multigraph_plot
(DetectedAI-plot), 32
frequency_vs_threshold_variable_multigraph_plot
(DetectedAI-plot), 32
frequency_vs_threshold_variable_plot
(DetectedAI-plot), 32
frequency_vs_threshold_variable_plot
(DetectedAI-plot), 32
frequency_vs_threshold_variable_plot, DetectedAI-method
(DetectedAI-plot), 32
frequency_vs_threshold_variable_plot, DetectedAI-class
(DetectedAI-plot), 32
frequency_vs_threshold_variable_plot, DetectedAI-method
(DetectedAI-plot), 32
frequency_vs_threshold_variable_summary
(DetectedAI-summary), 34
frequency_vs_threshold_variable_summary, DetectedAI-method
(DetectedAI-summary), 34
frequency_vs_threshold_variable_summary, DetectedAI-class
(DetectedAI-summary), 34
getAnnotationsFromAnnotation
(Annotation-wrappers), 4
getExonsFromAnnotation
(Annotation-wrappers), 4
getGenesFromAnnotation
(Annotation-wrappers), 4
getTranscriptsFromAnnotation
(Annotation-wrappers), 4
getTranscriptsVector
(Annotation-wrappers), 4
GlobalAnalysis-class (GlobalAnalysis-class), 44
GlobalAnalysis-class, 44
GlobalAnalysis-method
(GlobalAnalysis-class), 43
GlobalAnalysis-method
(GlobalAnalysis-class), 44
glocationplot
(ASEset-glocationplot), 16
glocationplot, ASEset-method
(ASEset-glocationplot), 16
GRvariants, 45, 71
hetFilt
(ASEset-filters), 14
hetFilt, ASEset-method
(ASEset-filters), 14
hist
(histplot), 45
hist, ASEset-method
(histplot), 45
hist, ReferenceBias-method
(histplot), 45
histplot, 45
impBamGAL
(import-bam), 47
impBamGAL, character-method
(import-bam), 47
impBamGRL
(import-bam-2), 48
impBcfGR
(import-bcf), 49
impBcfGR, character-method
(import-bcf), 49
implodeList.old, 46
import-bam, 47
import-bam-2, 48
import-bcf, 49
inferAlleles, 50
INDEX

reference_frequency_density_vs_threshold_variable_plot (DetectedAI-plot), 32
referenceFrequency (DetectedAI-class), 31
referenceFrequency, DetectedAI-method (DetectedAI-class), 31
refExist (ASEset-class), 9
refExist, ASEset-method (ASEset-class), 9
RegionSummary (RegionSummary-class), 73
regionSummary, ASEset-method (regionSummary), 72
regionSummary, numeric-method (regionSummary), 72
RegionSummary-class, 73
RegionSummary-method (RegionSummary-class), 73
RiskVariant (RiskVariant-class), 74
RiskVariant-class, 74
RiskVariant-method (RiskVariant-class), 74
RiskVariantFromGRangesAndPhaseArray (initialize-RiskVariant), 57
scanForHeterozygotes, 39, 49
scanForHeterozygotes (ASEset-scanForHeterozygotes), 20
scanForHeterozygotes, ASEset-method (ASEset-scanForHeterozygotes), 20
scanForHeterozygotes, GAlignmentsList-method (ASEset-scanForHeterozygotes), 20
scanForHeterozygotes.old, 75
sumnames (RegionSummary-class), 73
sumnames, RegionSummary-method (RegionSummary-class), 73

thresholdCountSample (DetectedAI-class), 31
thresholdCountSample, DetectedAI-method (DetectedAI-class), 31
thresholdDeltaFrequency (DetectedAI-class), 31
thresholdDeltaFrequency, DetectedAI-method (DetectedAI-class), 31
thresholdFrequency (DetectedAI-class), 31
thresholdFrequency, DetectedAI-method (DetectedAI-class), 31
thresholdPValue (DetectedAI-class), 31

usedSNPs_vs_threshold_variable_summary (DetectedAI-summary), 34
usedSNPs_vs_threshold_variable_summary, DetectedAI-method (DetectedAI-summary), 34

thresholdPValue (DetectedAI-class), 31