Package 'AllelicImbalance'

October 7, 2014

Type Package

Title Investigates allele specific expression

Version 1.2.0

Date 2014-03-25

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

Suggests
RUnit, org.Hs.eg.db,TxDb.Hsapiens.UCSC.hg19.knownGene,SNPlocs.Hsapiens.dbSNP.20120608

Depends GenomicAlignments, GenomicRanges, R (>= 3.0.0)

Imports methods, BiocGenerics, IRanges, Biostrings, Rsamtools,GenomicFeatures, AnnotationDbi

LazyData TRUE

Collate AllClasses.R AllGenerics.R initialize-methods.R
ASEset-accessors.R auxillary-functions.R chisq.test-methods.R
binom.test-methods.R barplot-methods.R locationplot-methods.R

biocViews Genetics, Infrastructure, Allelic Imbalance, AI, ASE,Sequencing

R topics documented:

`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:	2014-03-25
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 reagion 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

`ASEset-barplot` *barplot ASEset objects*

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

```
  ## S4 method for signature ASEset
barplot(
height,
type="count",
sampleColour=NULL,
legend=TRUE,
pValue=TRUE,
strand="nonStranded",
testValue=NULL,
testValue2=NULL,
OrgDb=NULL,
TxDb=NULL,
annotationType=c("gene","exon","transcript"),
main = NULL,
ylim=NULL,
yaxis=TRUE,
xaxis=FALSE,
ylab = TRUE,
xlab = TRUE,
legend.colnames = "",
las.ylab = 1,
las.xlab = 2,
cex.main = 0.9,
```

```
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,
verbose = FALSE,
...)
```

Arguments

height	An ASEset object
type	"count" or "fraction"
sampleColour	User specified colours
legend	Display legend
pValue	Display p-value
strand	Five options, "nonStranded","+", "-", "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 TranscriptDb 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
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
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.
verbose	Makes function more talkative
...	for simpler generics when extending function

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])
```

ASEset-class *ASEset objects*

Description

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

Details

An ASEset object differs from a regular SummarizedExperiment 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`.

Four different alleleCount options are available. The simples one is the nonStranded option, and is experiments where the strand information is not known. In this option both the plus, minus and unknown strand will be counted and present. The unknown strand is when the aligner could not find any strand associated with the read. 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.

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 `SummarizedExperiment` class.

Constructor

ASEsetFromCountList(rowData, countListNonStranded = NULL, countListPlus = NULL, countList-Minus = NULL, countListUnknown = NULL, colData = NULL, mapBiasExpMean = array(),verbose=FALSE ...)

Arguments:

rowData 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

Accessors

In the following code snippets, `x` is an ASEset object.

`alleleCounts(x, strand="nonStranded")`: Get SNP allele count list. Strand options are "non-Stranded", "+","-" or "*".

`mapBias(x)`: Get SNP mapping bias matrix.

`fraction(x, strand="nonStranded")`: Get SNP allele fraction matrix. strand options are "nonStranded", "+","-" or"*".

Subsetting

Subset like a `SummarizedExperiment`.

Author(s)

Jesper R. Gadin, Lasse Folkersen

See Also

- The `SummarizedExperiment` for ranges operations.

Examples

```
#make example countList
set.seed(42)
countListPlus <- list()
snps <- c("snp1","snp2","snp3","snp4","snp5")
for(snp in snps){
  count<-matrix(rep(0,20),ncol=5,dimnames=list(
c("sample1","sample2","sample3","sample4"),
c("A","T","G","C","del")))
  #insert random counts in two of the alleles
  for(allele in sample(c("A","T","G","C","del"),2)){
count[,allele]<-as.integer(rnorm(4,mean=50,sd=10))
  }
  countListPlus[[snp]] <- count
}

#make example rowData
rowData <- 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(rowData, countListPlus=countListPlus,
colData=colData)
```

ASEset-locationplot *locationplot ASEset objects*

Description

plotting ASE effects over a specific genomic region

Usage

```
## S4 method for signature ASEset
locationplot(
x,
type="fraction",
strand="nonStranded",
yaxis=TRUE,
xaxis=FALSE,
xlab=FALSE,
ylab=TRUE,
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,
...)
```

Arguments

x	an ASEset object.
type	"fraction" or "count"
strand	"+","-","*","both" or "nonStranded". 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
ylab	showing labels for the tic marks
xlab	showing labels for the tic marks
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 TranscriptDb object from which to plot an exon map.
verbose	Setting verbose=TRUE gives details of procedure during function run
...	arguments passed on to barplot function

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(rowData(ASEset))),])

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

`binom.test` *binomial test*

Description

Performs a binomial test on an ASEset object.

Usage

```
#snpAFTotDf comes from the mergeAFList function
## S4 method for signature ASEset
binom.test(x,n="nonStranded")
```

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.

Examples

```
#load example data
data(ASEset)

#make a binomial test
binom.test(ASEset,"nonStranded")
```

chisq.test *chi-square test*

Description

Performs a chisq.test on an ASEset object.

Usage

```
#snpAFTotDf comes from the mergeAFList function
## S4 method for signature ASEset
chisq.test(x,y="nonStranded")
```

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 nonStranded strand
chisq.test(ASEset)
```

funGetAnnotation-functions
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)
  getExonsFromAnnotation(TxDb,GR,leftFlank=0,rightFlank=0,
      verbose=FALSE)
  getTranscriptsFromAnnotation(TxDb,GR,leftFlank=0,
      rightFlank=0,verbose=FALSE)
  getCDSFromAnnotation(TxDb,GR,leftFlank=0,rightFlank=0,
    verbose=FALSE)

  getGenesVector(OrgDb, GR, leftFlank=0, rightFlank=0, verbose=FALSE)
  getExonsVector(TxDb, GR,leftFlank=0,rightFlank=0, verbose=FALSE)
  getTranscriptsVector(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
TxDb	A transcriptDb object
strand	Two options,"+" or "-"
annotationType	select one or more from "gene", "exon", "transcript", "cds".
verbose	A logical making the functions more talkative

Details

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

Value

The `getGenesFromAnnotation` function will return a `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

The `getExonsFromAnnotation` function will return a `GRanges object` with ranges over the exons in the region.

The `getTranscriptsFromAnnotation` function will return a `GRanges object` with ranges over the transcripts in the region.

The `getCDSFromAnnotation` function will return a `GRanges object` with ranges over the CDSFs in the region.

The `getExonsVector` function will return a character vector where each element are exons separated by comma

The `getTranscriptsVector` function will return a character vector where each element are transcripts separated by comma

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

```
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 <- rowData(ASEset)
#get annotation
g <- getGenesFromAnnotation(OrgDb,GR)
e <- getExonsFromAnnotation(TxDb,GR)
t <- getTranscriptsFromAnnotation(TxDb,GR)
c <- getCDSFromAnnotation(TxDb,GR)
```

`getAlleleCounts` *snp count data*

Description

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

Usage

```
getAlleleCounts(BamList, GRvariants, strand, return.type="list", verbose=TRUE)
```

Arguments

`BamList`	A `GAlignmentsList object` or `GRangesList object` containing data imported from a bam file
`GRvariants`	A `GRanges object` that contains positions of SNPs to retrieve
`strand`	A length 1 `character` with value 'nonStranded', '+', '-', or '*'. This argument determines if `getAlleleCounts` will retrieve counts from all reads, or only from reads marked as '+', '-' or '*' (unknown), respectively.
`return.type`	"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 SNPs. In the `strand` argument, specifying 'nonStranded' is the same as retrieving the sum count of '+' and '-' reads (and '*' unknown in case these are found in the bam file). 'nonStranded' is the default behaviour and can be used when the RNA-seq experiments strand information is not available.

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)

#set seqlevels in reads equal to seqlevels in GRvariants
seqlevels(reads) <- "17"

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

#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="-")
```

---

getAlleleCounts2 *snp count data*

Description

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

Usage

```
getAlleleCounts2(BamList, GRvariants, strand, verbose=TRUE)
```

Arguments

BamList	A GAlignmentsList object or GRangesList object containing data imported from a bam file
GRvariants	A GRanges object that contains positions of SNPs to retrieve
strand	A length 1 character with value 'nonStranded', '+', '-', or '*'. This argument determines if getAlleleCounts2 will retrieve counts from all reads, or only from reads marked as '+', '-' or '*' (unknown), respectively.
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 SNPs. In the strand argument, specifying 'nonStranded' is the same as retrieving the sum count of '+' and '-' reads (and '*' unknown in case these are found in the bam file). 'nonStranded' is the default behaviour and can be used when the RNA-seq experiments strand information is not available.

Value

getAlleleCounts2 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)

#set seqlevels in reads equal to seqlevels in GRvariants
seqlevels(reads) <- "17"

#get counts at the three positions specified in GRvariants
alleleCount <- getAlleleCounts2(BamList=reads,GRvariants,
strand="nonStranded")

#if the reads had contained stranded data, these two calls would
#have given the correct input objects for getAlleleCounts2
alleleCountPlus <- getAlleleCounts2(BamList=reads,GRvariants,
strand="+")
alleleCountMinus <- getAlleleCounts2(BamList=reads,GRvariants,
strand="-")
```

---

getAreaFromGeneNames *Get Gene Area*

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, 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
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

```
#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 list

Usage

```
getDefaultMapBiasExpMean(alleleCountList)
getDefaultMapBiasExpMean3D(alleleCountList)
```

Arguments

alleleCountList	A GRangesList object containing read information

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

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

getSnpIdFromLocation *Get rsIDs from locations of SNP*

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

```
getSnpIdFromLocation(GR, SNPloc, return.vector=FALSE, verbose=TRUE)
```

Arguments

GR	A GRanges that contains positions of SNPs to look up
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

Examples

```
#load example data
data(ASEset)

#get counts at the three positions specified in GRvariants
library(SNPlocs.Hsapiens.dbSNP.20120608)
updatedGRanges<-getSnpIdFromLocation(rowData(ASEset), SNPlocs.Hsapiens.dbSNP.20120608)
rowData(ASEset)<-updatedGRanges
```

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

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

import-bam-functions *Import Bam*

Description

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

Usage

```
impBamGAL(UserDir, searchArea, XStag = FALSE, verbose = TRUE)
impBamGRL(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
XStag	Setting XStag=TRUE stores the strand specific information in the mcols slot 'XS'
verbose	Setting verbose=TRUE gives details of procedure during function run.

Details

These functions are 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 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.

If the sequence data is strand-specific you may want to set XStag=TRUE. The strand specific information will then be stored in the meta columns with column name 'XS'.

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.

Note

A typical next step after the import of bam files is to obtain SNP information. This can be done either with the `impBcfGRL` and `getAlleleCounts` functions. Alternatively the `scanForHeterozygotes` function provides R-based functionality for identifying heterozygote coding SNPs.

Author(s)

Jesper R. Gadin, Lasse Folkersen

See Also

- The `impBcfGRL` for importing Bcf files.

Examples

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

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

reads <- impBamGAL(pathToFiles,searchArea,verbose=FALSE)
```

`import-bcf-functions` *Import Bcf Selection*

Description

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

Usage

```
impBcfGRL(UserDir, searchArea, verbose = TRUE)
impBcfGR(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 the procedure during function run.

Details

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

Value

`BcfImpGRList` 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

```
#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)
```

`initialize-ASEset` *Initialize ASEset*

Description

Functions to construct ASEset objects

Usage

```
ASEsetFromCountList(rowData, countListNonStranded = NULL, countListPlus =
                NULL, countListMinus = NULL, countListUnknown = NULL,
                colData = NULL, mapBiasExpMean = NULL, verbose =
                FALSE, ...)
```

Arguments

`rowData`	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` where 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

Details

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

countListNonStranded, countListPlus, countListMinus and countListUnknown are i.e. the outputs from the getAlleleCounts function.

Value

ASEsetFromCountList returns an ASEset object.

Note

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

Author(s)

Jesper R. Gadin, Lasse Folkersen

See Also

- The SummarizedExperiment for ranges operations.

Examples

```
#make example alleleCountListPlus
set.seed(42)
countListPlus <- list()
snps <- c("snp1","snp2","snp3","snp4","snp5")
for(snp in snps){
count<-matrix(rep(0,20),ncol=5,dimnames=list(
c("sample1","sample2","sample3","sample4"),
c("A","T","G","C","del")))
#insert random counts in two of the alleles
for(allele in sample(c("A","T","G","C","del"),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(0,20),ncol=5,dimnames=list(
c("sample1","sample2","sample3","sample4"),
c("A","T","G","C","del")))
#insert random counts in two of the alleles
for(allele in sample(c("A","T","G","C","del"),2)){
count[,allele]<-as.integer(rnorm(4,mean=50,sd=10))
}
```

```
countListMinus[[snp]] <- count
}


#make example rowData
rowData <- 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(rowData, countListPlus=countListPlus,
countListMinus=countListMinus, colData=colData)
```

`reads` *reads object*

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 Folkersen

References

Montgomery SB et al. Transcriptome genetics using second generation sequencing in a Caucasian population. Nature. 2010 Apr 1;464(7289):773-7.

See Also

- The `GRvariants` which is another example object

Examples

```
##load eample data (Not Run)
#data(reads)
```

scanForHeterozygotes *scanForHeterozygotes*

Description

Identifies the positions of SNPs found in BamGR reads.

Usage

```
scanForHeterozygotes(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 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

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

Index