Package ‘ChIPpeakAnno’

May 21, 2024

Type  Package
Title  Batch annotation of the peaks identified from either ChIP-seq, ChIP-chip experiments, or any experiments that result in large number of genomic interval data
Version  3.38.0
Encoding  UTF-8
Author  Lihua Julie Zhu,
        Jianhong Ou,
        Jun Yu,
        Kai Hu,
        Haibo Liu,
        Junhui Li,
        Hervé Pagès,
        Claude Gazin,
        Nathan Lawson,
        Ryan Thompson,
        Simon Lin,
        David Lapointe,
        Michael Green
Maintainer  Jianhong Ou <jianhong.ou@duke.edu>,
            Lihua Julie Zhu <julie.zhu@umassmed.edu>,
            Kai Hu <kai.hu@umassmed.edu>,
            Junhui Li <junhui.li@umassmed.edu>
Depends  R (>= 3.5), methods, IRanges (>= 2.13.12), GenomicRanges (>= 1.31.8), S4Vectors (>= 0.17.25)
Imports  AnnotationDbi, BiocGenerics (>= 0.1.0), Biostrings (>= 2.47.6), pwalign, DBI, dplyr, GenomeInfoDb, GenomicAlignments, GenomicFeatures, RBGL, Rsamtools, SummarizedExperiment, VennDiagram, biomaRt, ggplot2, grDevices, graph, graphics, grid, InteractionSet, KEGGREST, matrixStats, multiTest, regioneR, rtracklayer, stats, utils, universalmotif, stringr, tibble, tidyR, data.table, scales, ensembldb
Suggests  AnnotationHub, BSgenome, limma, reactome.db, BiocManager, BiocStyle, BSgenome.Ecoli.NCBI.20080805,
BSgenome.Hsapiens.UCSC.hg19, org.Ce.eg.db, org.Hs.eg.db,
BSgenome.Celegans.UCSC.ce10, BSgenome.Drerio.UCSC.danRer7,
BSgenome.Hsapiens.UCSC.hg38, DelayedArray, idr, seqinr,
EnsDb.Hsapiens.v75, EnsDb.Hsapiens.v79, EnsDb.Hsapiens.v86,
TxDb.Hsapiens.UCSC.hg18.knownGene,
TxDb.Hsapiens.UCSC.hg19.knownGene,
TxDb.Hsapiens.UCSC.hg38.knownGene, GO.db, gplots, UpSetR,
knitr, rmarkdown, reshape2, testthat, trackViewer, motifStack,
OrganismDbi, BiocFileCache

**Description**  The package encompasses a range of functions for identifying the
closest gene, exon, miRNA, or custom features—such as highly conserved
elements and user-supplied transcription factor binding sites.
Additionally, users can retrieve sequences around the peaks and obtain
enriched Gene Ontology (GO) or Pathway terms. In version 2.0.5 and beyond,
new functionalities have been introduced. These include features for
identifying peaks associated with bi-directional promoters along with
summary statistics (peaksNearBDP), summarizing motif occurrences in
peaks (summarizePatternInPeaks), and associating additional identifiers
with annotated peaks or enrichedGO (addGeneIDs). The package integrates
with various other packages such as bioMaRt, IRanges, Biostrings, BSgenome,
GO.db, multtest, and stat to enhance its analytical capabilities.

**License**  GPL (>= 2)

**LazyLoad**  yes

**LazyData**  true

**LazyDataCompression**  xz

**biocViews**  Annotation, ChIPSeq, ChIPchip

**VignetteBuilder**  knitr

**RoxygenNote**  7.3.1

**git_url**  https://git.bioconductor.org/packages/ChIPpeakAnno

**git_branch**  RELEASE_3_19

**git_last_commit**  26e03e3

**git_last_commit_date**  2024-04-30

**Repository**  Bioconductor 3.19

**Date/Publication**  2024-05-21

**Contents**

- ChIPpeakAnno-package ............................................... 4
- addAncestors .......................................................... 6
- addGeneIDs ............................................................ 7
- addMetadata ........................................................... 9
- annoGR-class ........................................................ 10
- annoPeaks ............................................................... 11
### Index

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>oligoSummary</td>
<td>82</td>
</tr>
<tr>
<td>peakPermTest</td>
<td>83</td>
</tr>
<tr>
<td>Peaks.Ste12.Replicate1</td>
<td>85</td>
</tr>
<tr>
<td>Peaks.Ste12.Replicate2</td>
<td>86</td>
</tr>
<tr>
<td>Peaks.Ste12.Replicate3</td>
<td>86</td>
</tr>
<tr>
<td>peaks1</td>
<td>87</td>
</tr>
<tr>
<td>peaks2</td>
<td>88</td>
</tr>
<tr>
<td>peaks3</td>
<td>88</td>
</tr>
<tr>
<td>peaksNearBDP</td>
<td>89</td>
</tr>
<tr>
<td>permPool-class</td>
<td>90</td>
</tr>
<tr>
<td>pie1</td>
<td>91</td>
</tr>
<tr>
<td>plotBinOverRegions</td>
<td>93</td>
</tr>
<tr>
<td>preparePool</td>
<td>94</td>
</tr>
<tr>
<td>reCenterPeaks</td>
<td>95</td>
</tr>
<tr>
<td>summarizeOverlapsByBins</td>
<td>96</td>
</tr>
<tr>
<td>summarizePatternInPeaks</td>
<td>97</td>
</tr>
<tr>
<td>tileCount</td>
<td>99</td>
</tr>
<tr>
<td>tileGRanges</td>
<td>100</td>
</tr>
<tr>
<td>toGRanges</td>
<td>101</td>
</tr>
<tr>
<td>translatePattern</td>
<td>104</td>
</tr>
<tr>
<td>TSS.human.GRCh37</td>
<td>105</td>
</tr>
<tr>
<td>TSS.human.GRCh38</td>
<td>105</td>
</tr>
<tr>
<td>TSS.human.NCBI36</td>
<td>106</td>
</tr>
<tr>
<td>TSS.mouse.GRCm38</td>
<td>107</td>
</tr>
<tr>
<td>TSS.mouse.NCBIM37</td>
<td>107</td>
</tr>
<tr>
<td>TSS.rat.RGSC3.4</td>
<td>108</td>
</tr>
<tr>
<td>TSS.rat.Rnor_5.0</td>
<td>109</td>
</tr>
<tr>
<td>TSS.zebrafish.Zv8</td>
<td>109</td>
</tr>
<tr>
<td>TSS.zebrafish.Zv9</td>
<td>110</td>
</tr>
<tr>
<td>TxDb2GR</td>
<td>111</td>
</tr>
<tr>
<td>wgEncodeTfbsV3</td>
<td>111</td>
</tr>
<tr>
<td>write2FASTA</td>
<td>113</td>
</tr>
<tr>
<td>xget</td>
<td>114</td>
</tr>
</tbody>
</table>

**ChIPpeakAnno-package**  
*Batch annotation of the peaks identified from either ChIP-seq or ChIP-chip experiments.*

### Description

The package includes functions to retrieve the sequences around the peak, obtain enriched Gene Ontology (GO) terms, find the nearest gene, exon, miRNA or custom features such as most conserved elements and other transcription factor binding sites leveraging biomaRt, IRanges, Biostrings, BSgenome, GO.db, hypergeometric test phyper and multtest package.
Details

Package: ChIPpeakAnno
Type: Package
Version: 3.0.0
Date: 2014-10-24
License: LGPL
LazyLoad: yes

Author(s)

Lihua Julie Zhu, Jianhong Ou, Hervé Pagès, Claude Gazin, Nathan Lawson, Simon Lin, David Lapointe and Michael Green

Maintainer: Jianhong Ou <jianhong.ou@umassmed.edu>, Lihua Julie Zhu <julie.zhu@umassmed.edu>

References

4. S. Dudoit, J. P. Shaffer, and J. C. Boldrick (Submitted). Multiple hypothesis testing in microarray experiments.

Examples

```r
if(interactive()){  
data(myPeakList)
library(enssembldb)
library(EnsDb.Hsapiens.v75)
anno <- annoGR(EnsDb.Hsapiens.v75)
annotatedPeak <-
  annotatePeakInBatch(myPeakList[1:6], AnnotationData=anno)
```
addAncestors

Add GO IDs of the ancestors for a given vector of GO ids

Description

Add GO IDs of the ancestors for a given vector of GO IDs leveraging GO.db

Usage

addAncestors(go.ids, ontology = c("bp", "cc", "mf"))

Arguments

- go.ids: A matrix with 4 columns: first column is GO IDs and 4th column is entrez IDs.
- ontology: bp for biological process, cc for cellular component and mf for molecular function.

Value

A vector of GO IDs containing the input GO IDs with the GO IDs of their ancestors added.

Author(s)

Lihua Julie Zhu

Examples

go.ids = cbind(c("GO:0008150", "GO:0005576", "GO:0003674"),
               c("ND", "IDA", "ND"),
               c("BP", "BP", "BP"),
               c("1", "1", "1"))
library(GO.db)
addAncestors(go.ids, ontology="bp")
**addGeneIDs**

Add common IDs to annotated peaks such as gene symbol, entrez ID, ensemble gene id and refseq id.

**Description**
Add common IDs to annotated peaks such as gene symbol, entrez ID, ensemble gene id and refseq id leveraging organism annotation dataset. For example, org.Hs.eg.db is the dataset from orgs.Hs.eg.db package for human, while org.Mm.eg.db is the dataset from the org.Mm.eg.db package for mouse.

**Usage**

```r
addGeneIDs(
  annotatedPeak,
  orgAnn,
  IDs2Add = c("symbol"),
  feature_id_type = "ensembl_gene_id",
  silence = TRUE,
  mart
)
```

**Arguments**

- **annotatedPeak**: GRanges or a vector of feature IDs.
- **orgAnn**: organism annotation dataset such as org.Hs.eg.db.
- **IDs2Add**: a vector of annotation identifiers to be added
- **feature_id_type**: type of ID to be annotated, default is ensembl_gene_id
- **silence**: TRUE or FALSE. If TRUE, will not show unmapped entrez id for feature ids.
- **mart**: mart object, see useMart of biomaRt package for details

**Details**

One of orgAnn and mart should be assigned.

- If orgAnn is given, parameter feature_id_type should be ensemble_gene_id, entrez_id, gene_symbol, gene_alias or refseq_id. And parameter IDs2Add can be set to any combination of identifiers such as "accnum", "ensembl", "ensemblprot", "ensembltrans", "entrez_id", "enzyme", "genename", "pfam", "pmid", "prosite", "refseq", "symbol", "unigene" and "uniprot". Some IDs are unique to an organism, such as "omim" for org.Hs.eg.db and "mgi" for org.Mm.eg.db.

Here is the definition of different IDs:

- accnum: GenBank accession numbers
- ensembl: Ensembl gene accession numbers
- ensemblprot: Ensembl protein accession numbers

•
addGeneIDs

- ensembltrans: Ensembl transcript accession numbers
- entrez_id: entrez gene identifiers
- enzyme: EC numbers
- genename: gene name
- pfam: Pfam identifiers
- pmid: PubMed identifiers
- prosite: PROSITE identifiers
- refseq: RefSeq identifiers
- symbol: gene abbreviations
- unigene: UniGene cluster identifiers
- uniprot: Uniprot accession numbers
- omim: OMIM(Mendelian Inheritance in Man) identifiers
- mgi: Jackson Laboratory MGI gene accession numbers

- If mart is used instead of orgAnn, for valid parameter feature_id_type and IDs2Add parameters, please refer to getBM in bioMart package. Parameter feature_id_type should be one valid filter name listed by listFilters(mart) such as ensemble_gene_id. And parameter IDs2Add should be one or more valid attributes name listed by listAttributes(mart) such as external_gene_id, entrezgene, wikigene_name, or mirbase_transcript_name.

Value

GRanges if the input is a GRanges or dataframe if input is a vector.

Author(s)

Jianhong Ou, Lihua Julie Zhu

References

http://www.bioconductor.org/packages/release/data/annotation/

See Also

getBM, AnnotationDb

Examples

data(annotatedPeak)
library(org.Hs.eg.db)
addGeneIDs(annotatedPeak[1:6,],orgAnn="org.Hs.eg.db",
   IDs2Add=c("symbol","omim"))
##addGeneIDs(annotatedPeak$feature[1:6],orgAnn="org.Hs.eg.db",
##   IDs2Add=c("symbol","genename"))
if(interactive()){
   mart <- useMart("ENSEMBL_MART_ENSEMBL",host="www.ensembl.org",
      dataset="hsapiens_gene_ensembl")
   ##mart <- useMart(biomart="ensembl",dataset="hsapiens_gene_ensembl")
   addGeneIDs(annotatedPeak[1:6,], mart=mart,"}
addMetadata

Add metadata of the GRanges objects used for findOverlapsOfPeaks

Description

Add metadata to to overlapping peaks after calling findOverlapsOfPeaks.

Usage

addMetadata(ol, colNames = NULL, FUN = c, ...)

Arguments

ol An object of overlappingPeaks, which is output of findOverlapsOfPeaks.

colNames Names of metadata column to be added. If it is NULL, addMetadata will guess what to add.

FUN A function to be called

... Arguments to the function call.

Value

return value is An object of overlappingPeaks.

Author(s)

Jianhong Ou

See Also

See Also as findOverlapsOfPeaks

Examples

peaks1 <- GRanges(seqnames=c(6,6,6,6,5),
  IRanges(start=c(1543200,1557200,1563000,1569800,167889600),
         end=c(1555199,1560599,1565199,1573799,167893599),
         names=c("p1","p2","p3","p4","p5")),
  strand="+",
  score=1:5, id=letters[1:5])

peaks2 <- GRanges(seqnames=c(6,6,6,6,5),
  IRanges(start=c(1549800,1554400,1565000,1569400,16788000),
         end=c(1555199,1560599,1565199,1573799,16789000),
         names=c("f1","f2","f3","f4","f5")),
  strand="+",
  score=6:10, id=LETTERS[1:5])


annoGR-class

Class annoGR

Description

An object of class annoGR represents the annotation data could be used by annotationPeakInBatch.

Usage

## S4 method for signature 'annoGR'
info(object)

## S4 method for signature 'GRanges'
annoGR(ranges, feature = "group", date, ...)

## S4 method for signature 'TxDb'
annoGR(
  ranges,
  feature = c("gene", "transcript", "exon", "CDS", "fiveUTR", "threeUTR", "microRNA", "tRNAs", "geneModel"),
  date,
  source,
  mdata,
  OrganismDb
)

## S4 method for signature 'EnsDb'
annoGR(
  ranges,
  feature = c("gene", "transcript", "exon", "disjointExons"),
  date,
  source,
  mdata
)

Arguments

object annoGR object.
ranges an object of GRanges, TxDb or EnsDb
feature annotation type
date a Date object
... could be following parameters
annoPeaks

source character, where the annotation comes from
mdata data frame, metadata from annotation
OrganismDb an object of OrganismDb. It is used for extracting gene symbol for geneModel group for TxDb

Slots

seqnames, ranges, strand, elementMetadata, seqinfo slots inherit from GRanges. The ranges must have unique names.
source character, where the annotation comes from
date a Date object
feature annotation type, could be "gene", "exon", "transcript", "CDS", "fiveUTR", "threeUTR", "microRNA", "tRNAs", "geneModel" for TxDb object, or "gene", "exon", "transcript" for EnsDb object
mdata data frame, metadata from annotation

Objects from the Class

Objects can be created by calls of the form new("annoGR", date, elementMetadata, feature, mdata, ranges, seqinfo, seqnames, source, strand)

Author(s)

Jianhong Ou

Examples

if(interactive() || Sys.getenv("USER")=="jianhongou"){
  library(EnsDb.Hsapiens.v79)
  anno <- annoGR(EnsDb.Hsapiens.v79)
}

annoPeaks  Annotate peaks

Description

Annotate peaks by annoGR object in the given range.
annoPeaks

Usage

annoPeaks(
  peaks,
  annoData,
  bindingType = c("nearestBiDirectionalPromoters", "startSite", "endSite", "fullRange"),
  bindingRegion = c(-5000, 5000),
  ignore.peak.strand = TRUE,
  select = c("all", "bestOne"),
  ...
)

Arguments

peaks                  peak list, GRanges object
annoData              annotation data, GRanges object
bindingType           Specifying the criteria to associate peaks with annotation. Here is how to use it together with the parameter bindingRegion
  • To obtain peaks within 5kb upstream and up to 3kb downstream of TSS within the gene body, set bindingType = "startSite" and bindingRegion = c(-5000, 3000)
  • To obtain peaks up to 5kb upstream within the gene body and 3kb downstream of gene/Exon End, set bindingType = "endSite" and bindingRegion = c(-5000, 3000)
  • To obtain peaks from 5kb upstream to 3kb downstream of genes/Exons, set bindingType = "fullRange" and bindingRegion = c(-5000, 3000)
  • To obtain peaks with nearest bi-directional promoters within 5kb upstream and 3kb downstream of TSS, set bindingType = "nearestBiDirectionalPromoters" and bindingRegion = c(-5000, 3000)
startSite             start position of the feature (strand is considered)
endSite               end position of the feature (strand is considered)
fullRange             whole range of the feature
nearestBiDirectionalPromoters nearest promoters from both direction of the peaks (strand is considered). It will report bidirectional promoters if there are promoters in both directions in the given region (defined by bindingRegion). Otherwise, it will report the closest promoter in one direction.
bindingRegion         Annotation range used together with bindingType, which is a vector with two integer values, default to c (-5000, 5000). The first one must be no bigger than 0, which means upstream. And the second one must be no less than 1, which means downstream (1 is the site position, 2 is the next base of the site position). For details, see bindingType.
ignore.peak.strand    ignore the peaks strand or not.
select                "all" or "bestOne". Return the annotation containing all or the best one. The "bestOne" is selected by the shortest distance to the sites and then similarity between peak and annotations. Ignored if bindingType is nearestBiDirectionalPromoters.
annotatedPeak

... Not used.

Value

Output is a GRanges object of the annotated peaks.

Author(s)

Jianhong Ou

See Also

See Also as annotatePeakInBatch

Examples

library(ensembldb)
library(EnsDb.Hsapiens.v75)
data("myPeakList")
annoGR <- toGRanges(EnsDb.Hsapiens.v75)
seqlevelsStyle(myPeakList) <- seqlevelsStyle(annoGR)
annoPeaks(myPeakList, annoGR)

annotatedPeak  Annotated Peaks

Description

TSS annotated putative STAT1-binding regions that are identified in un-stimulated cells using ChIP-seq technology (Robertson et al., 2007)

Usage

annotatedPeak

Format

GRanges with slot start holding the start position of the peak, slot end holding the end position of the peak, slot names holding the id of the peak, slot strand holding the strands and slot space holding the chromosome location where the peak is located. In addition, the following variables are included.

list("feature") id of the feature such as ensembl gene ID
list("insideFeature") upstream: peak resides upstream of the feature; downstream: peak resides downstream of the feature; inside: peak resides inside the feature; overlapStart: peak overlaps with the start of the feature; overlapEnd: peak overlaps with the end of the feature; includeFeature: peak include the feature entirely
list("distancetoFeature") distance to the nearest feature such as transcription start site
annotatePeakInBatch

- `list("start_position")` start position of the feature such as gene
- `list("end_position")` end position of the feature such as the gene

Details

- obtained by data(TSS.human.GRCh37)
- data(myPeakList)
- annotatePeakInBatch(myPeakList, AnnotationData = TSS.human.GRCh37, output="b", multiple=F)

Examples

```r
data(annotatedPeak)
head(annotatedPeak, 4)  # show first 4 ranges
if (interactive() || Sys.getenv("USER") == "jianhongou") {
  y = annotatedPeak$distancetoFeature[!is.na(annotatedPeak$distancetoFeature)]
  hist(as.numeric(as.character(y)),  
       xlab="Distance To Nearest TSS", main="", breaks=1000,  
       ylim=c(0, 50), xlim=c(min(as.numeric(as.character(y)))-100,  
       max(as.numeric(as.character(y)))+100))
}
```

---

**annotatePeakInBatch** *Obtain the distance to the nearest TSS, miRNA, and/or exon for a list of peaks*

**Description**

Obtain the distance to the nearest TSS, miRNA, exon et al for a list of peak locations leveraging IRanges and biomaRt package

**Usage**

```r
annotatePeakInBatch(  
  myPeakList,  
  mart,  
  featureType = c("TSS", "miRNA", "Exon"),  
  AnnotationData,  
  output = c("nearestLocation", "overlapping", "both", "shortestDistance", "inside",  
    "upstream&inside", "inside&downstream", "upstream", "downstream",  
    "upstreamORdownstream", "nearestBiDirectionalPromoters"),  
  multiple = c(TRUE, FALSE),  
  maxgap = -1L,  
  PeakLocForDistance = c("start", "middle", "end", "endMinusStart"),  
  FeatureLocForDistance = c("TSS", "middle", "start", "end", "geneEnd"),  
  select = c("all", "first", "last", "arbitrary"),  
  ignore.strand = TRUE,
)```
bindingRegion = NULL,
...
)

### Arguments

- **myPeakList**: A GRanges object
- **mart**: A mart object, used if AnnotationData is not supplied, see useMart of bioMaRt package for details
- **featureType**: A character vector used with mart argument if AnnotationData is not supplied; choose from "TSS", "miRNA" or "Exon"
- **AnnotationData**: A GRanges or annoGR object. It can be obtained from the function getAnnotation or customized annotation of class GRanges containing additional variable: strand (1 or + for plus strand and -1 or - for minus strand). Pre-compiled annotations, such as TSS.human.NCBI36, TSS.mouse.NCBIM37, TSS.rat.RGSC3.4 and TSS.zebrafish.Zv8, are provided by this package (attach them with data() function). Another method to provide annotation data is to obtain through biomaRt in real time by using the mart and featureType option
- **output**: 
  - nearestLocation (default) will output the nearest features calculated as PeakLoc - FeatureLocForDistance; when selected, the output can consist of both "strictly nearest features (non-overlapping)" and "overlapping features" as long as they are the nearest
  - overlapping will output overlapping features with maximum gap specified as maxgap between peak range and feature range; it is possible for a peak to be annotated with zero ("NA" will be returned) or multiple overlapping features if exist
  - both will output all the nearest features as well as any features that overlap with the peak that is not the nearest
  - shortestDistance will output the features with the shortest distance; the "shortest distance" is determined from either ends of the feature to either ends of the peak
  - upstream&inside will output all upstream and overlapping features with maximum gap
  - inside&downstream will output all downstream and overlapping features with maximum gap
  - upstream will output all upstream features with maximum gap
  - downstream will output all downstream features with maximum gap
  - upstreamORdownstream will output all upstream features with maximum gap or downstream with maximum gap
  - nearestBiDirectionalPromoters will use annoPeaks to annotate peaks. Nearest promoters from both direction of the peaks (strand is considered). It will report bidirectional promoters if there are promoters in both directions in the given region (defined by bindingRegion). Otherwise, it will report the closest promoter in one direction.
- **multiple**: Not applicable when output is nearest. TRUE: output multiple overlapping features for each peak. FALSE: output at most one overlapping feature for each peak. This parameter is kept for backward compatibility, please use select.
annotatePeakInBatch

maxgap
The maximum gap that is allowed between 2 ranges for the ranges to be considered as overlapping. The gap between 2 ranges is the number of positions that separate them. The gap between 2 adjacent ranges is 0. By convention when one range has its start or end strictly inside the other (i.e. non-disjoint ranges), the gap is considered to be -1.

PeakLocForDistance
Specify the location of peak for calculating distance, i.e., middle means using middle of the peak to calculate distance to feature, start means using start of the peak to calculate the distance to feature, endMinusStart means using the end of the peak to calculate the distance to features on plus strand and the start of the peak to calculate the distance to features on minus strand. To be compatible with previous version, by default using start.

FeatureLocForDistance
Specify the location of feature for calculating distance, i.e., middle means using middle of the feature to calculate distance of peak to feature, start means using start of the feature to calculate the distance to feature, TSS means using start of feature when feature is on plus strand and using end of feature when feature is on minus strand, geneEnd means using end of feature when feature is on plus strand and using start of feature when feature is on minus strand. To be compatible with previous version, by default using TSS.

select
"all" may return multiple overlapping peaks, "first" will return the first overlapping peak, "last" will return the last overlapping peak and "arbitrary" will return one of the overlapping peaks.

ignore.strand
When set to TRUE, the strand information is ignored in the annotation. Unless you have stranded peaks and you are interested in annotating peaks to the features in the same strand only, you should just use the default setting ignore.strand = TRUE.

bindingRegion
Annotation range used for annoPeaks, which is a vector with two integer values, default to c(-5000, 5000). The first one must be no bigger than 0. And the second one must be no less than 1. Once bindingRegion is defined, annotation will based on annoPeaks. Here is how to use it together with the parameter output and FeatureLocForDistance.

- To obtain peaks with nearest bi-directional promoters within 5kb upstream and 3kb downstream of TSS, set output = "nearestBiDirectionalPromoters" and bindingRegion = c(-5000, 3000)
- To obtain peaks within 5kb upstream and up to 3kb downstream of TSS within the gene body, set output="overlapping", FeatureLocForDistance="TSS" and bindingRegion = c(-5000, 3000)
- To obtain peaks up to 5kb upstream within the gene body and 3kb downstream of gene/Exon End, set output="overlapping", FeatureLocForDistance="geneEnd" and bindingRegion = c(-5000, 3000)
- To obtain peaks from 5kb upstream to 3kb downstream of genes/Exons, set output="overlapping", bindingType = "fullRange" and bindingRegion = c(-5000, 3000)

For details, see annoPeaks.

Parameters could be passed to annoPeaks.
Value

An object of GRanges with slot start holding the start position of the peak, slot end holding the end position of the peak, slot space holding the chromosome location where the peak is located, slot rownames holding the id of the peak. In addition, the following variables are included.

list("feature")
id of the feature such as ensembl gene ID

list("insideFeature")
upstream: peak resides upstream of the feature; downstream: peak resides downstream of the feature; inside: peak resides inside the feature; overlapStart: peak overlaps with the start of the feature; overlapEnd: peak overlaps with the end of the feature; includeFeature: peak include the feature entirely

list("distancetoFeature")
distance to the nearest feature such as transcription start site. By default, the distance is calculated as the distance between the start of the binding site and the TSS that is the gene start for genes located on the forward strand and the gene end for genes located on the reverse strand. The user can specify the location of peak and location of feature for calculating this

list("start_position")
start position of the feature such as gene

list("end_position")
end position of the feature such as the gene

list("strand") 1 or + for positive strand and -1 or - for negative strand where the feature is located

list("shortestDistance")
The shortest distance from either end of peak to either end the feature.

list("fromOverlappingOrNearest")
Relevant only when output is set to "both". If "nearestLocation": indicates this feature’s start (feature’s end for features from minus strand) is the closest to the peak start ("strictly nearest" or "nearest overlapping"); if "Overlapping": indicates this feature overlaps with this peak although it is not the nearest (non-nearest overlapping)

Author(s)

Lihua Julie Zhu, Jianhong Ou

References


See Also

getAnnotation, findOverlappingPeaks, makeVennDiagram, addGeneIDs, peaksNearBDP, summarizePatternInPeaks, annoGR, annoPeaks
Examples

```r
## example 1: annotate myPeakList by TxDb or EnsDb.
library(ensmbldb)
library(EnsDb.Hsapiens.v75)
annoData <- annoGR(EnsDb.Hsapiens.v75)
annotatePeak = annotatePeakInBatch(myPeakList[1:6], AnnotationData=annoData)
annotatePeak

## example 2: annotate myPeakList (GRanges) with TSS.human.NCBI36 (Granges)
data(TSS.human.NCBI36)
annotatedPeak = annotatePeakInBatch(myPeakList[1:6],
                                         AnnotationData=TSS.human.NCBI36)
annotatedPeak

## example 3: you have a list of transcription factor binding sites from
## literature and are interested in determining the extent of the overlap
## to the list of peaks from your experiment. Prior calling the function
## annotatePeakInBatch, need to represent both dataset as GRanges
## where start is the start of the binding site, end is the end of the
## binding site, names is the name of the binding site, space and strand
## are the chromosome name and strand where the binding site is located.

myexp <- GRanges(seqnames=c(6,6,6,6,5,4,4),
                 IRanges(start=c(1543200,1557200,1563000,1569800,
                                167889600,100,1000),
                           end=c(1555199,1560599,1565199,1573799,
                                167893599,200,1200),
                           names=c("p1","p2","p3","p4","p5","p6","p7"),
                                             strand="+"))
literature <- GRanges(seqnames=c(6,6,6,6,5,4,4),
                      IRanges(start=c(1549800,1554400,1565000,1569400,
                                     167888600,120,800),
                             end=c(1550599,1560799,1565399,1571199,
                                  167888999,140,1400),
                             names=c("f1","f2","f3","f4","f5","f6","f7")),
                             strand=rep("+", "-", c(5, 2)))
annotatedPeak1 <- annotatePeakInBatch(myexp,
                                      AnnotationData=literature)
pie(table(annotatedPeak1$insideFeature))
annotatedPeak1

## use toGRanges or rtracklayer::import to convert BED or GFF format
## to GRanges before calling annotatePeakInBatch

## use the data.frame to create the BED format
start = c("100", "1000"),
end = c("200", "1100"),
name = c("peak1", "peak2")
```

library(testthat)

peak <- GRanges(seqnames = "chr1",
  IRanges(start = 24736757, end=24737528,
    names = "testPeak"))

data(TSS.human.GRCh37)

TSS.human.GRCh37[names(TSS.human.GRCh37)== "ENSG00000001461"]
# GRanges object with 1 range and 1 metadata column:
#   names ranges strand | description
#<Rle> <IRanges> <Rle> | <character>
# ENSG00000001461 1 24742285-24799466 + | NIPA-like domain con..

peak
#GRanges object with 1 range and 0 metadata columns:
#   ranges strand
#<Rle> <IRanges> <Rle>
# testPeak chr1 24736757-24737528 *

TSS.human.GRCh37[names(TSS.human.GRCh37)== "ENSG00000001460"]
#GRanges object with 1 range and 1 metadata column:
#   ranges strand | description
#<Rle> <IRanges> <Rle> | <character>
# ENSG00000001460 1 24683490-24743424 - | UPF0490 protein C1or..

ap <- annotatePeakInBatch(peak, Annotation=TSS.human.GRCh37,
  PeakLocForDistance = "start")
stopifnot(ap$feature=="ENSG00000001461")

ap <- annotatePeakInBatch(peak, Annotation=TSS.human.GRCh37,
  PeakLocForDistance = "end")
stopifnot(ap$feature=="ENSG00000001461")

ap <- annotatePeakInBatch(peak, Annotation=TSS.human.GRCh37,
  PeakLocForDistance = "middle")
stopifnot(ap$feature=="ENSG00000001461")

ap <- annotatePeakInBatch(peak, Annotation=TSS.human.GRCh37,
  PeakLocForDistance = "endMinusStart")
stopifnot(ap$feature=="ENSG00000001461")

## Let’s calculate the distances between the peak and the TSS of the genes
## in the annotation file used for annotating the peaks.
## Please note that we need to compute the distance using the annotation
## file TSS.human.GRCh37.
## If you would like to use TxDb.Hsapiens.UCSC.hg19.knownGene,
## then you will need to annotate the peaks
## using TxDb.Hsapiens.UCSC.hg19.knownGene as well.
### using start
start(peak) -start(TSS.human.GRCh37[names(TSS.human.GRCh37)==
  "ENSG00000001461"])
# [1] -5528
start(peak) -end(TSS.human.GRCh37[names(TSS.human.GRCh37)==
  "ENSG00000001460"])
# [1] -6667

### using middle
(start(peak) + end(peak))/2 -
  start(TSS.human.GRCh37[names(TSS.human.GRCh37)==
    "ENSG00000001461"])
# [1] -5142.5

(start(peak) + end(peak))/2 -
  end(TSS.human.GRCh37[names(TSS.human.GRCh37)==
    "ENSG00000001460"])
# [1] 49480566
annotatePeakInBatch

end(peak) - start(TSS.human.GRCh37[names(TSS.human.GRCh37) == 
  "ENSG00000001461"])) # picked
# [1] -4757
end(peak) - end(TSS.human.GRCh37[names(TSS.human.GRCh37) == 
  "ENSG00000001460"])
# [1] -5896

### using endMinusStart
end(peak) - start(TSS.human.GRCh37[names(TSS.human.GRCh37) == 
  "ENSG00000001461"])) ## picked
# [1] -4575
start(peak) - end(TSS.human.GRCh37[names(TSS.human.GRCh37) == 
  "ENSG00000001460"])
# [1] -6667

##### using txdb object to annotate the peaks
library(org.Hs.eg.db)
select(org.Hs.eg.db, key="STPG1", keytype="SYMBOL",
columns=c("ENSEMBL", "ENTREZID", "SYMBOL"))
# SYMBOL  ENSEMBL  ENTREZID
# STPG1   ENSG00000001460  90529
select(org.Hs.eg.db, key="ENSG00000001461", keytype="ENSEMBL",
columns=c("ENSEMBL", "ENTREZID", "SYMBOL"))
# ENSEMBL  ENTREZID  SYMBOL
# ENSG00000001461  57185  NIPAL3
require(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb.ann <- genes(TxDB.Hsapiens.UCSC.hg19.knownGene)
STPG1 <- select(org.Hs.eg.db, key="STPG1", keytype="SYMBOL",
columns=c("SYMBOL", "ENSEMBL", "ENTREZID"))[1,3]
NIPAL3 <- select(org.Hs.eg.db, key="NIPAL3", keytype="SYMBOL",
columns=c("SYMBOL", "ENSEMBL", "ENTREZID"))[1,3]
ap <- annotatePeakInBatch(peak, Annotation=txdb.ann,
  PeakLocForDistance = "start")
expect_equal(ap$feature, STPG1)
ap <- annotatePeakInBatch(peak, Annotation=txdb.ann,
  PeakLocForDistance = "end")
expect_equal(ap$feature, STPG1)
ap <- annotatePeakInBatch(peak, Annotation=txdb.ann,
  PeakLocForDistance = "middle")
expect_equal(ap$feature, STPG1)
ap <- annotatePeakInBatch(peak, Annotation=txdb.ann,
  PeakLocForDistance = "endMinusStart")
expect_equal(ap$feature, NIPAL3)
txdb.ann[NIPAL3]
txdb.ann[,txdb.ann$gene_id == NIPAL3]
# GRanges object with 1 range and 1 metadata column:
# seqnames ranges strand | gene_id
# <Rle> <IRanges> <Rle> | <character>
# 57185 chr1 24742245-24799473  + | 57185
#-------
txdb.ann[,txdb.ann$gene_id == STPG1]
# GRanges object with 1 range and 1 metadata column:
# seqnames ranges strand | gene_id
# <Rle> <IRanges> <Rle> | <character>
# 90529 chr1 24683489-24741587  - | 90529
assignChromosomeRegion

Summarize peak distribution over exon, intron, enhancer, proximal promoter, 5 prime UTR and 3 prime UTR

Description

Summarize peak distribution over exon, intron, enhancer, proximal promoter, 5 prime UTR and 3 prime UTR

Usage

assignChromosomeRegion(
  peaks.RD,
  exon,
  TSS,
  utr5,
  utr3,
  proximal.promoter.cutoff = c(upstream = 2000, downstream = 100),
  immediate.downstream.cutoff = c(upstream = 0, downstream = 1000),
  nucleotideLevel = FALSE,
  precedence = NULL,
  TxDb = NULL
)

Arguments

peaks.RD peaks in GRanges: See example below
exon exon data obtained from getAnnotation or customized annotation of class GRanges containing additional variable: strand (1 or + for plus strand and -1 or - for minus strand). This parameter is for backward compatibility only. TxDb should be used instead.
TSS TSS data obtained from getAnnotation or customized annotation of class GRanges containing additional variable: strand (1 or + for plus strand and -1 or - for minus strand). For example, data(TSS.human.NCBI36), data(TSS.mouse.NCBIM37), data(TSS.rat.RGSC3.4) and data(TSS.zebrafish.Zv8). This parameter is for backward compatibility only. TxDb should be used instead.
utr5 5 prime UTR data obtained from getAnnotation or customized annotation of class GRanges containing additional variable: strand (1 or + for plus strand and -1 or - for minus strand). This parameter is for backward compatibility only. TxDb should be used instead.
utr3 3 prime UTR data obtained from getAnnotation or customized annotation of class GRanges containing additional variable: strand (1 or + for plus strand and -1 or - for minus strand). This parameter is for backward compatibility only. TxDb should be used instead.
proximal.promoter.cutoff
Specify the cutoff in bases to classify proximal promoter or enhancer. Peaks that reside within proximal.promoter.cutoff upstream from or overlap with transcription start site are classified as proximal promoters. Peaks that reside upstream of the proximal.promoter.cutoff from gene start are classified as enhancers. The default is upstream 2000 bases and downstream 100 bases.

immediate.downstream.cutoff
Specify the cutoff in bases to classify immediate downstream region or enhancer region. Peaks that reside within immediate.downstream.cutoff downstream of gene end but not overlap 3' UTR are classified as immediate downstream. Peaks that reside downstream over immediate.downstream.cutoff from gene end are classified as enhancers. The default is upstream 0 bases and downstream 1000 bases.

nucleotideLevel
Logical. Choose between peak centric and nucleotide centric view. Default=FALSE

precedence
If no precedence specified, double count will be enabled, which means that if a peak overlap with both promoter and 5' UTR, both promoter and 5' UTR will be incremented. If a precedence order is specified, for example, if promoter is specified before 5' UTR, then only promoter will be incremented for the same example. The values could be any combinations of "Promoters", "immediateDownstream", "fiveUTRs", "threeUTRs", "Exons" and "Introns", Default=NULL

TxDb
an object of TxDb

Value
A list of two named vectors: percentage and jaccard (Jaccard Index). The information in the vectors:

list("Exons") Percent or the picard index of the peaks resided in exon regions.
list("Introns") Percent or the picard index of the peaks resided in intron regions.
list("fiveUTRs") Percent or the picard index of the peaks resided in 5 prime UTR regions.
list("threeUTRs") Percent or the picard index of the peaks resided in 3 prime UTR regions.
list("Promoter") Percent or the picard index of the peaks resided in proximal promoter regions.
list("ImmediateDownstream") Percent or the picard index of the peaks resided in immediate downstream regions.
list("Intergenic.Region") Percent or the picard index of the peaks resided in intergenic regions.

The Jaccard index, also known as Intersection over Union. The Jaccard index is between 0 and 1. The higher the index, the more significant the overlap between the peak region and the genomic features in consideration.

Author(s)
Jianhong Ou, Lihua Julie Zhu
References


See Also

genomicElementDistribution, genomicElementUpSetR, binOverFeature, binOverGene, binOverRegions

Examples

if (interactive() || Sys.getenv("USER")=='jianhongou'){  
    ##Display the list of genomes available at UCSC:  
    #library(rtracklayer)  
    #ucscGenomes()[, "db"]  
    ## Display the list of Tracks supported by makeTxDbFromUCSC()  
    #supportedUCSCtables()  
    ##Retrieving a full transcript dataset for Human from UCSC  
    ##TranscriptDb <-  
    ##      makeTxDbFromUCSC(genome="hg19", tablename="ensGene")  
    if(require(TxDb.Hsapiens.UCSC.hg19.knownGene)){  
        TxDb <- TxDb.Hsapiens.UCSC.hg19.knownGene  
        exons <- exons(TxDB, columns=NULL)  
        fiveUTRs <- unique(unlist(fiveUTRsByTranscript(TxDB)))  
        Feature.distribution <-  
            assignChromosomeRegion(exons, nucleotideLevel=TRUE, TxDb=TxDB)  
        barplot(Feature.distribution$percentage)  
        assignChromosomeRegion(fiveUTRs, nucleotideLevel=FALSE, TxDb=TxDB)  
        data(myPeakList)  
        assignChromosomeRegion(myPeakList, nucleotideLevel=TRUE,  
            precedence=c("Promoters", "immediateDownstream",  
                "fiveUTRs", "threeUTRs",  
                "Exons", "Introns"),  
            TxDb=TxDB)  
    }  

}  

bdp obtain the peaks near bi-directional promoters

Description

Obtain the peaks near bi-directional promoters. Also output percent of peaks near bi-directional promoters.
Usage

bdp(peaks, annoData, maxgap = 2000L, ...)

Arguments

peaks       peak list, *GRanges* object
annoData   annotation data, *annoGR* object
maxgap     maxgap between peak and TSS
...        Not used.

Value

Output is a list of *GRanges* object of the peaks near bi-directional promoters.

Author(s)

Jianhong Ou

See Also

See Also as *annoPeaks*, *annoGR*

Examples

```r
if(interactive() || Sys.getenv("USER")="jianhongou"){
  library(ensembldb)
  library(EnsDb.Hsapiens.v75)
  data("myPeakList")
  annoGR <- annoGR(EnsDb.Hsapiens.v75)
  seqlevelsStyle(myPeakList) <- seqlevelsStyle(annoGR)
  ChIPpeakAnno:::bdp(myPeakList, annoGR)
}
```

bindist-class

Class "bindist"

Description

An object of class "bindist" represents the relevant fixed-width range of binding site from the feature and number of possible binding site in each range.

Objects from the Class

Objects can be created by calls of the form `new("bindist", counts="integer", mids="integer", halfBinSize="integer", bindingType="character", featureType="character")`. 
binOverFeature

See Also

preparePool, peakPermTest

---

**binOverFeature**  
*Aggregate peaks over bins from the TSS*

**Description**

Aggregate peaks over bins from the feature sites.

**Usage**

```r
binOverFeature(
  ..., annotationData = GRanges(),
  select = c("all", "nearest"),
  radius = 5000L,
  nbins = 50L,
  minGeneLen = 1L,
  aroundGene = FALSE,
  mbins = nbins,
  featureSite = c("FeatureStart", "FeatureEnd", "bothEnd"),
  PeakLocForDistance = c("all", "end", "start", "middle"),
  FUN = sum,
  errFun = sd,
  xlab,
  ylab,
  main
)
```

**Arguments**

- `...`: Objects of GRanges to be analyzed
- `annotationData`: An object of GRanges or annoGR for annotation
- `select`: Logical: annotate the peaks to all features or the nearest one
- `radius`: The radius of the longest distance to feature site
- `nbins`: The number of bins
- `minGeneLen`: The minimal gene length
- `aroundGene`: Logical: count peaks around features or a given site of the features. Default = FALSE
- `mbins`: if aroundGene set as TRUE, the number of bins intra-feature. The value will be normalized by `value * (radius/genelen) * (mbins/nbins)`
- `featureSite`: which site of features should be used for distance calculation
binOverGene

PeakLocForDistance
which site of peaks should be used for distance calculation

FUN
the function to be used for score calculation

erFun
the function to be used for errorbar calculation or values for the errorbar.

tax
xlab
titles for each x axis

ty
ylab
titles for each y axis

main
overall titles for each plot

Value
A data.frame with bin values.

Author(s)
Jianhong Ou

Examples

```r
bed <- system.file("extdata", "MACS_output.bed", package="ChIPpeakAnno")
gr1 <- toGRanges(bed, format="BED", header=FALSE)
data(TSS.human.GRCh37)
binOverFeature(gr1, annotationData=TSS.human.GRCh37, 
               radius=5000, nbins=10, FUN=length, errFun=0)
```

```
binOverGene

coverage of gene body

Description
calculate the coverage of gene body per gene per bin.

Usage

```r
binOverGene(
  cvglists,
  TxDb, 
  upstream.cutoff = 0L, 
  downstream.cutoff = upstream.cutoff, 
  nbinsGene = 100L, 
  nbinsUpstream = 20L, 
  nbinsDownstream = nbinsUpstream, 
  includeIntron = FALSE, 
  minGeneLen = nbinsGene, 
  maxGeneLen = Inf
)
```
binOverRegions

calculate the coverage of 5’UTR, CDS and 3’UTR per transcript per bin.

**Arguments**

- `cvglists`: A list of `SimpleRleList` or `RleList`. It represents the coverage for samples.
- `TxDb`: An object of `TxDb`. It is used for extracting the annotations.
- `upstream.cutoff, downstream.cutoff`: cutoff length for upstream or downstream of transcript.
- `nbinsGene, nbinsUpstream, nbinsDownstream`: The number of bins for gene, upstream and downstream.
- `includeIntron`: A logical value which indicates including intron or not.
- `minGeneLen, maxGeneLen`: minimal or maximal length of gene.

**Author(s)**

Jianhong Ou

**See Also**

- `binOverRegions`, `plotBinOverRegions`

**Examples**

```r
if(Sys.getenv("USER") == "jianhongou"){
  path <- system.file("extdata", package="ChIPpeakAnno")
  library(TxDb.Hsapiens.UCSC.hg19.knownGene)
  library(rtracklayer)
  files <- dir(path, "bigWig")
  if(.Platform$OS.type != "windows"){
    cvglists <- lapply(file.path(path, files), import,
                       format="BigWig", as="RleList")
    names(cvglists) <- sub(".bigWig", "", files)
    d <- binOverGene(cvglists, TxDb.Hsapiens.UCSC.hg19.knownGene)
    plotBinOverRegions(d)
  }
}
```

**Description**

- `coverage of chromosome regions`
Usage

```
binOverRegions(
  cvglists,
  TxDb,
  upstream.cutoff = 1000L,
  downstream.cutoff = upstream.cutoff,
  nbinsCDS = 100L,
  nbinsUTR = 20L,
  nbinsUpstream = 20L,
  nbinsDownstream = nbinsUpstream,
  includeIntron = FALSE,
  minCDSLen = nbinsCDS,
  minUTRLen = nbinsUTR,
  maxCDSLen = Inf,
  maxUTRLen = Inf
)
```

Arguments

- `cvglists`: A list of `SimpleRleList` or `RleList`. It represents the coverage for samples.
- `TxDb`: An object of `TxDb`. It is used for extracting the annotations.
- `upstream.cutoff`, `downstream.cutoff`: Cutoff length for upstream or downstream of transcript.
- `nbinsCDS`, `nbinsUTR`, `nbinsUpstream`, `nbinsDownstream`: The number of bins for CDS, UTR, upstream and downstream.
- `includeIntron`: A logical value which indicates including intron or not.
- `minCDSLen`, `minUTRLen`: Minimal length of CDS or UTR of transcript.
- `maxCDSLen`, `maxUTRLen`: Maximal length of CDS or UTR of transcript.

Author(s)

Jianhong Ou

See Also

`binOverGene`, `plotBinOverRegions`

Examples

```
if(Sys.getenv("USER") == "jianhongou"){
  path <- system.file("extdata", package="ChIPpeakAnno")
  library(TxDb.Hsapiens.UCSC.hg19.knownGene)
  library(rtracklayer)
  files <- dir(path, "bigWig")
  if(.Platform$OS.type != "windows"){
    cvglists <- lapply(file.path(path, files), import,
```
format="BigWig", as="RleList")
names(cvglists) <- sub("\.*\.bigWig", "", files)
d <- binOverRegions(cvglists, TxDb.Hsapiens.UCSC.hg19.knownGene)
plotBinOverRegions(d)
}

ChIPpeakAnno-deprecated

Deprecated Functions in Package ChIPpeakAnno

Description

These functions are provided for compatibility with older versions of R only, and may be defunct as soon as the next release.

Arguments

Peaks1          GRanges: See example below.
Peaks2          GRanges: See example below.
maxgap, minoverlap

Used in the internal call to findOverlaps() to detect overlaps. See ?findOverlaps in the IRanges package for a description of these arguments.

multiple

TRUE or FALSE: TRUE may return multiple overlapping peaks in Peaks2 for one peak in Peaks1; FALSE will return at most one overlapping peaks in Peaks2 for one peak in Peaks1. This parameter is kept for backward compatibility, please use select.

NameOfPeaks1    Name of the Peaks1, used for generating column name.
NameOfPeaks2    Name of the Peaks2, used for generating column name.
select

all may return multiple overlapping peaks, first will return the first overlapping peak, last will return the last overlapping peak and arbitrary will return one of the overlapping peaks.

annotate

Include overlapFeature and shortestDistance in the OverlappingPeaks or not. 1 means yes and 0 means no. Default to 0.

ignore.strand

When set to TRUE, the strand information is ignored in the overlap calculations.

connectedPeaks

If multiple peaks involved in overlapping in several groups, set it to "merge" will count it as only 1, while set it to "min" will count it as the minimal involved peaks in any concered groups

...            Objects of GRanges: See also findOverlapsOfPeaks.

Details

findOverlappingPeaks is now deprecated wrappers for findOverlapsOfPeaks
cntOverlaps \hspace{1cm} count overlaps

### Description

Count overlaps with max gap.

### Usage

```r
cntOverlaps(A, B, maxgap = 0L, ...)
```

### Arguments

- **A, B**: A GRanges object.
- **maxgap**: A single integer \( \geq 0 \).
- **...**: parameters passed to `numOverlaps`

### condenseMatrixByColnames

*Condense matrix by colnames*

### Description

Condense matrix by colnames

### Usage

```r
condenseMatrixByColnames(mx, iname, sep = ";", cnt = FALSE)
```

### Arguments

- **mx**: a matrix to be condensed
- **iname**: the name of the column to be condensed
- **sep**: separator for condensed values, default `;`
- **cnt**: TRUE/FALSE specifying whether adding count column or not?

### Value

dataframe of condensed matrix
Author(s)

Jianhong Ou, Lihua Julie Zhu

Examples

```r
c<-matrix(c(rep(rep(1:5,2),2),rep(1:10,2)),ncol=4)
colnames(c)<-c("con.1","con.2","index.1","index.2")
condenseMatrixByColnames(c,"con.1")
condenseMatrixByColnames(c,2)
```

---

**convert2EntrezID**

*Convert other common IDs to entrez gene ID.*

Description

Convert other common IDs such as ensemble gene id, gene symbol, refseq id to entrez gene ID leveraging organism annotation dataset. For example, org.Hs.eg.db is the dataset from orgs.Hs.eg.db package for human, while org.Mm.eg.db is the dataset from the org.Mm.eg.db package for mouse.

Usage

```r
convert2EntrezID(IDs, orgAnn, ID_type = "ensembl_gene_id")
```

Arguments

- **IDs**
  - a vector of IDs such as ensembl gene ids
- **orgAnn**
  - organism annotation dataset such as org.Hs.eg.db
- **ID_type**
  - type of ID: can be ensemble_gene_id, gene_symbol or refseq_id

Value

vector of entrez ids

Author(s)

Lihua Julie Zhu

Examples

```r
ensemblIDs = c("ENSG00000115956", "ENSG00000071082", "ENSG00000071054", "ENSG00000115594", "ENSG00000115594", "ENSG00000115594", "ENSG00000115598", "ENSG000001170417")
library(org.Hs.eg.db)
entrezIDs = convert2EntrezID(IDs=ensemblIDs, orgAnn="org.Hs.eg.db", ID_type="ensembl_gene_id")
```
countPatternInSeqs  "Output total number of patterns found in the input sequences"

Description
Output total number of patterns found in the input sequences

Usage
```
countPatternInSeqs(pattern, sequences)
```

Arguments
```
  pattern       DNAstringSet object
  sequences     a vector of sequences
```

Value
Total number of occurrence of the pattern in the sequences

Author(s)
Lihua Julie Zhu

See Also
summarizePatternInPeaks, translatePattern

Examples
```
library(Biostrings)
filepath = system.file("extdata", "examplePattern.fa", package="ChIPpeakAnno")
dict = readDNAStringSet(filepath = filepath, format="fasta", use.names=TRUE)
sequences = c("ACTGGGGGGGGCCTGGGCCCCCAAAT",
               "AAAAAACCCCTTTTGGCCATCCCGGGACGGGCCCAT",
               "ATCGAAAATTTCC")
countPatternInSeqs(pattern=dict[1], sequences=sequences)
countPatternInSeqs(pattern=dict[2], sequences=sequences)
pattern = DNAStringSet("ATNGMAA")
countPatternInSeqs(pattern=pattern, sequences=sequences)
```
cumulativePercentage

Plot the cumulative percentage tag allocation in sample

Description

Plot the difference between the cumulative percentage tag allocation in paired samples.

Usage

cumulativePercentage(bamfiles, gr, input = 1, binWidth = 1000, ...)

Arguments

bamfiles Bam file names.
gr An object of GRanges
input Which file name is input. default 1.
binWidth The width of each bin.
... parameter for summarizeOverlaps.

Value

A list of data.frame with the cumulative percentages.

Author(s)

Jianhong Ou

References


Examples

## Not run:
path <- system.file("extdata", "reads", package="MMDiffBamSubset")
files <- dir(path, "bam$", full.names = TRUE)
library(BSgenome.Hsapiens.UCSC.hg19)
gr <- as(seqinfo(Hsapiens)["chr1"], "GRanges")
cumulativePercentage(files, gr)

## End(Not run)
downstreams

*Get downstream coordinates*

**Description**

Returns an object of the same type and length as x containing downstream ranges. The output range is defined as

**Usage**

downstreams(gr, upstream, downstream)

**Arguments**

- `gr`: A GenomicRanges object
- `upstream`, `downstream`: non-negative integers.

**Details**

- \((\text{end}(x) - \text{upstream}) \text{ to } (\text{end}(x) + \text{downstream} - 1)\)
  for ranges on the + and * strand, and as
- \((\text{start}(x) - \text{downstream} + 1) \text{ to } (\text{start}(x) + \text{downstream})\)
  for ranges on the - strand.

Note that the returned object might contain out-of-bound ranges.

**Value**

A GenomicRanges object

**Examples**

gr <- GRanges("chr1", IRanges(rep(10, 3), width=6), c("+", "-", "*"))
downstreams(gr, 2, 2)
egOrgMap

Convert between the name of the organism annotation package ("OrgDb") and the name of the organism.

Description

Give a species name and return the organism annotation package name or give an organism annotation package name then return the species name.

Usage

egOrgMap(name)

Arguments

name The name of the organism annotation package or the species.

Value

A object of character

Author(s)

Jianhong Ou

Examples

egOrgMap("org.Hs.eg.db")
egOrgMap("Mus musculus")

enrichedGO Enriched Gene Ontology terms used as example

Description

Enriched Gene Ontology terms used as example

Usage

enrichedGO
Format

A list of 3 dataframes.

`list("bp")` dataframe described the enriched biological process with 9 columns
- `go.id`: GO biological process id
- `go.term`: GO biological process term
- `go.Definition`: GO biological process description
- `Ontology`: Ontology branch, i.e. BP for biological process
- `count.InDataset`: count of this GO term in this dataset
- `count.InGenome`: count of this GO term in the genome
- `pvalue`: pvalue from the hypergeometric test
- `totaltermInDataset`: count of all GO terms in this dataset
- `totaltermInGenome`: count of all GO terms in the genome

`list("mf")` dataframe described the enriched molecular function with the following 9 columns
- `go.id`: GO molecular function id
- `go.term`: GO molecular function term
- `go.Definition`: GO molecular function description
- `Ontology`: Ontology branch, i.e. MF for molecular function
- `count.InDataset`: count of this GO term in this dataset
- `count.InGenome`: count of this GO term in the genome
- `pvalue`: pvalue from the hypergeometric test
- `totaltermInDataset`: count of all GO terms in this dataset
- `totaltermInGenome`: count of all GO terms in the genome

`list("cc")` dataframe described the enriched cellular component the following 9 columns
- `go.id`: GO cellular component id
- `go.term`: GO cellular component term
- `go.Definition`: GO cellular component description
- `Ontology`: Ontology type, i.e. CC for cellular component
- `count.InDataset`: count of this GO term in this dataset
- `count.InGenome`: count of this GO term in the genome
- `pvalue`: pvalue from the hypergeometric test
- `totaltermInDataset`: count of all GO terms in this dataset
- `totaltermInGenome`: count of all GO terms in the genome

Author(s)

Lihua Julie Zhu

Examples

```r
data(enrichedGO)
dim(enrichedGO$mf)
dim(enrichedGO$cc)
dim(enrichedGO$bp)
```
enrichmentPlot

Description

Plot the GO/KEGG/reactome enrichment results

Usage

enrichmentPlot(
  res,
  n = 20,
  strlength = Inf,
  style = c("v", "h"),
  label_wrap = 40,
  label_substring_to_remove = NULL,
  orderBy = c("pvalue", "termId", "none")
)

Arguments

res output of getEnrichedGO, getEnrichedPATH.
n number of terms to be plot.
strlength shorten the description of term by the number of char.
style plot vertically or horizontally
label_wrap soft wrap the labels (i.e. descriptions of the GO or PATHWAY terms), default to 40 characters.
label_substring_to_remove remove common substring from label, default to NULL. Special characters must be escaped. E.g. if you would like to remove "Homo sapiens (human)" from labels, you must use "Homo sapiens \( human\)".
orderBy order the data by pvalue, termId or none.

Value

an object of ggplot

Author(s)

Jianhong Ou, Kai Hu
Examples

```r
data(enrichedGO)
enrichmentPlot(enrichedGO)
if (interactive()||Sys.getenv("USER")=="jianhongou") {
  library(org.Hs.eg.db)
  library(GO.db)
  bed <- system.file("extdata", "MACS_output.bed", package="ChIPpeakAnno")
  gr1 <- toGRanges(bed, format="BED", header=FALSE)
  gff <- system.file("extdata", "GFF_peaks.gff", package="ChIPpeakAnno")
  gr2 <- toGRanges(gff, format="GFF", header=FALSE, skip=3)
  library(EnsDb.Hsapiens.v75) ##(hg19)
  annoData <- toGRanges(EnsDb.Hsapiens.v75)
  gr1.anno <- annoPeaks(gr1, annoData)
  gr2.anno <- annoPeaks(gr2, annoData)
  over <- lapply(GRangesList(gr1=gr1.anno, gr2=gr2.anno),
                  getEnrichedGO, orgAnn="org.Hs.eg.db",
                  maxP=.05, minGOterm=10, condense=TRUE)
  enrichmentPlot(over$gr1)
  enrichmentPlot(over$gr2, style = "h")
}
```

---

**EnsDb2GR**  
*EnsDb object to GRanges*

**Description**

convert EnsDb object to GRanges

**Usage**

EnsDb2GR(ranges, feature)

**Arguments**

- **ranges**: an EnsDb object
- **feature**: feature type, could be disjointExons, gene, exon and transcript

**estFragmentLength**  
*estimate the fragment length*

**Description**

estimate the fragment length for bam files
estFragmentLength

Usage

estFragmentLength(
  bamfiles,
  index = bamfiles,
  plot = TRUE,
  lag.max = 1000,
  minFragmentSize = 100,
  ...
)

Arguments

bamfiles The file names of the 'BAM' ('SAM' for asBam) files to be processed.
index The names of the index file of the 'BAM' file being processed; this is given without the '.bai' extension.
plot logical. If TRUE (the default) the acf is plotted.
lag.max maximum lag at which to calculate the acf. See acf
minFragmentSize minimal fragment size to avoid the phantom peak.
... Not used.

Value

numeric vector

Author(s)

Jianhong Ou

Examples

if(interactive() || Sys.getenv("USER") == "jianhongou"){
  path <- system.file("extdata", "reads", package="MMdiffBamSubset")
  if(file.exists(path)){
    WT.AB2 <- file.path(path, "WT_2.bam")
    Null.AB2 <- file.path(path, "Null_2.bam")
    Resc.AB2 <- file.path(path, "Resc_2.bam")
    estFragmentLength(c(WT.AB2, Null.AB2, Resc.AB2))
  }
}
estLibSize  

estimate the library size

Description

estimate the library size of bam files

Usage

estLibSize(bamfiles, index = bamfiles, ...)

Arguments

bamfiles  The file names of the 'BAM' ('SAM' for asBam) files to be processed.
index    The names of the index file of the 'BAM' file being processed; this is given without the '.bai' extension.
...      Not used.

Value

numeric vector

Author(s)

Jianhong Ou

Examples

if(interactive() || Sys.getenv("USER") == "jianhongou"){
  path <- system.file("extdata", "reads", package="MMDiffBamSubset")
  if(file.exists(path)){
    WT.AB2 <- file.path(path, "WT_2.bam")
    Null.AB2 <- file.path(path, "Null_2.bam")
    Resc.AB2 <- file.path(path, "Resc_2.bam")
    estLibSize(c(WT.AB2, Null.AB2, Resc.AB2))
  }
}


**Description**

Gene model with exon, 5' UTR and 3' UTR information for human sapiens (GRCh37) obtained from biomaRt

**Usage**

ExonPlusUtr.human.GRCh37

**Format**

GRanges with slot start holding the start position of the exon, slot end holding the end position of the exon, slot rownames holding ensembl transcript id and slot space holding the chromosome location where the gene is located. In addition, the following variables are included.

- `list("strand")` 1 for positive strand and -1 for negative strand
- `list("description")` description of the transcript
- `list("ensembl_gene_id")` gene id
- `list("utr5start")` 5' UTR start
- `list("utr5end")` 5' UTR end
- `list("utr3start")` 3' UTR start
- `list("utr3end")` 3' UTR end

**Details**

used in the examples Annotation data obtained by: mart = useMart(biomart = "ensembl", dataset = "hsapiens_gene_ensembl") ExonPlusUtr.human.GRCh37 = getAnnotation(mart=human, feature TypeName="ExonPlusUtr")

**Examples**

data(ExonPlusUtr.human.GRCh37)
slotNames(ExonPlusUtr.human.GRCh37)
featureAlignedDistribution

plot distribution in given ranges

Description

plot distribution in the given feature ranges

Usage

featureAlignedDistribution(
  cvglists,
  feature.gr,
  upstream,
  downstream,
  n.tile = 100,
  zeroAt,
  ...
)

Arguments

cvglists Output of featureAlignedSignal or a list of SimpleRleList or RleList
feature.gr An object of GRanges with identical width. If the width equal to 1, you can use upstream and downstream to set the range for plot. If the width not equal to 1, you can use zeroAt to set the zero point of the heatmap.
upstream, downstream
  upstream or downstream from the feature.gr.
n.tile The number of tiles to generate for each element of feature.gr, default is 100
zeroAt zero point position of feature.gr
...
  any parameters could be used by matplot

Value

invisible matrix of the plot.

Author(s)

Jianhong Ou

See Also

See Also as featureAlignedSignal, featureAlignedHeatmap
featureAlignedExtendSignal

Examples

cvglists <- list(A=RleList(chr1=Rle(sample.int(5000, 100),
            sample.int(300, 100))),
  B=RleList(chr1=Rle(sample.int(5000, 100),
            sample.int(300, 100))))
feature.gr <- GRanges("chr1", IRanges(seq(1, 4900, 100), width=100))
featureAlignedDistribution(cvglists, feature.gr, zeroAt=50, type="l")

Description

eXtract signals in the given feature ranges from bam files (DNAseq only). The reads will be extended
to estimated fragment length.

Usage

featureAlignedExtendSignal(
  bamfiles,  
  index = bamfiles,  
  feature.gr, 
  upstream, 
  downstream, 
  n.tile = 100, 
  fragmentLength, 
  librarySize, 
  pe = c("auto", "PE", "SE"), 
  adjustFragmentLength, 
  gal, 
  ...  
)

Arguments

bamfiles The file names of the 'BAM' ('SAM' for asBam) files to be processed.
index The names of the index file of the 'BAM' file being processed; this is given
without the '.bai' extension.
feature.gr An object of GRanges with identical width.
upstream, downstream 
  upstream or downstream from the feature.gr.
n.tile The number of tiles to generate for each element of feature.gr, default is 100
fragmentLength Estimated fragment length.
librarySize Estimated library size.
pe Pair-end or not. Default auto.
adjustFragmentLength
   A numeric vector with length 1. Adjust the fragments/reads length to.
gal A GAlignmentsList object or a list of GAlignmentPairs. If bamfiles is missing, gal is required.
... Not used.

Value
A list of matrix. In each matrix, each row record the signals for corresponding feature.

Author(s)
Jianhong Ou

See Also
See Also as `featureAlignedSignal`, `estLibSize`, `estFragmentLength`

Examples
```r
if(interactive() || Sys.getenv("USER")="jianhongou"){
  path <- system.file("extdata", package="MMDiffBamSubset")
  if(file.exists(path)){
    WT.AB2 <- file.path(path, "reads", "WT_2.bam")
    Null.AB2 <- file.path(path, "reads", "Null_2.bam")
    Resc.AB2 <- file.path(path, "reads", "Resc_2.bam")
    peaks <- file.path(path, "peaks", "WT_2_Macs_peaks.xls")
    estLibSize(c(WT.AB2, Null.AB2, Resc.AB2))
    feature.gr <- toGRanges(peaks, format="MACS")
    feature.gr <- feature.gr[seqnames(feature.gr)="chr1" &
                            start(feature.gr)>3000000 &
                            end(feature.gr)<75000000]
    sig <- featureAlignedExtendSignal(c(WT.AB2, Null.AB2, Resc.AB2),
                                       feature.gr=reCenterPeaks(feature.gr, width=1),
                                       upstream = 505,
                                       downstream = 505,
                                       n.tile=101,
                                       fragmentLength=250,
                                       librarySize=1e9)
    featureAlignedHeatmap(sig, reCenterPeaks(feature.gr, width=1010),
                           zeroAt=.5, n.tile=101)
  }
}
```
featureAlignedHeatmap  

Heatmap representing signals in given ranges

Description
plot heatmap in the given feature ranges

Usage
featureAlignedHeatmap(
  cvglists,
  feature.gr,
  upstream,
  downstream,
  zeroAt,
  n.tile = 100,
  annoMcols = c(),
  sortBy = names(cvglists)[1],
  color = colorRampPalette(c("yellow", "red")(50),
  lower.extreme,
  upper.extreme,
  margin = c(0.1, 0.01, 0.15, 0.1),
  gap = 0.01,
  newpage = TRUE,
  gp = gpar(fontsize = 10),
  ...
)

Arguments

cvglists  Output of featureAlignedSignal or a list of SimpleRleList or RleList
feature.gr  An object of GRanges with identical width. If the width equal to 1, you can use upstream and downstream to set the range for plot. If the width not equal to 1, you can use zeroAt to set the zero point of the heatmap.
upstream, downstream  upstream or downstream from the feature.gr. It must keep same as featureAlignedSignal. It is used for x-axis label.
zeroAt  zero point position of feature.gr
n.tile  The number of tiles to generate for each element of feature.gr, default is 100
annoMcols  The columns of metadata of feature.gr that specifies the annotations shown on the right side of the heatmap.
sortBy  Sort the feature.gr by columns by annoMcols and then the signals of the given samples. Default is the first sample. Set to NULL to disable sort.
color  vector of colors used in heatmap
lower.extreme, upper.extreme  
   The lower and upper boundary value of each samples
margin   Margin for of the plot region.
gap      Gap between each heatmap columns.
newpage  Call grid.newpage or not. Default, TRUE
gp       A gpar object can be used for text.
...      Not used.

Value
   invisible gList object.

Author(s)
   Jianhong Ou

See Also
   See Also as featureAlignedSignal, featureAlignedDistribution

Examples
   cvglists <- list(A=RleList(chr1=Rle(sample.int(5000, 100),  
                         sample.int(300, 100))),  
                 B=RleList(chr1=Rle(sample.int(5000, 100),  
                           sample.int(300, 100))))
   feature.gr <- GRanges("chr1", IRanges(seq(1, 4900, 100), width=100))
   feature.gr$anno <- rep(c("type1", "type2"), c(25, 24))
   featureAlignedHeatmap(cvglists, feature.gr, zeroAt=50, annoMcols="anno")

featureAlignedSignal  extract signals in given ranges

Description
   extract signals in the given feature ranges

Usage
   featureAlignedSignal(  
      cvglists,  
      feature.gr,  
      upstream,  
      downstream,  
      n.tile = 100,  
      ...  
   )
**findEnhancers**

Find possible enhancers depend on DNA interaction data.

### Description

Find possible enhancers by data from chromosome conformation capture techniques such as 3C, 5C or HiC.

### Arguments

- **cvglits**: List of `SimpleRleList` or `RleList`.
- **feature.gr**: An object of `GRanges` with identical width.
- **upstream, downstream**: Set the feature.gr to upstream and downstream from the center of the feature.gr if they are set.
- **n.tile**: The number of tiles to generate for each element of feature.gr, default is 100.
- **...**: Not used.

### Value

A list of matrix. In each matrix, each row record the signals for corresponding feature. rownames of the matrix show the seqnames and coordinates.

### Author(s)

Jianhong Ou

### See Also

See Also as `featureAlignedHeatmap`, `featureAlignedDistribution`

### Examples

```r
cvglits <- list(A=RleList(chr1=Rle(sample.int(5000, 100),
sample.int(300, 100))),
                B=RleList(chr1=Rle(sample.int(5000, 100),
sample.int(300, 100))))
feature.gr <- GRanges("chr1", IRanges(seq(1, 4900, 100), width=100))
featureAlignedSignal(cvglits, feature.gr)
```
findEnhancers

Usage

findEnhancers(
  peaks,
  annoData,
  DNAinteractiveData,
  bindingType = c("nearestBiDirectionalPromoters", "startSite", "endSite"),
  bindingRegion = c(-5000, 5000),
  ignore.peak.strand = TRUE,
  ...
)

Arguments

peaks  peak list, GRanges object
annoData  annotation data, GRanges object
DNAinteractiveData  DNA interaction data, GRanges object with interaction blocks informations, GInteractions object, or BEDPE file which could be imported by importGInteractions or BiocIO::import or assembly in following list: hg38, hg19, mm10, danRer10, danRer11.

bindingType  Specifying the criteria to associate peaks with annotation. Here is how to use it together with the parameter bindingRegion. The annotation will be shift to a new position depend on the DNA interaction region.
  • To obtain peaks within 5kb upstream and up to 3kb downstream of shift TSS within the gene body, set bindingType = "startSite" and bindingRegion = c(-5000, 3000)
  • To obtain peaks up to 5kb upstream within the gene body and 3kb downstream of shift gene/Exon End, set bindingType = "endSite" and bindingRegion = c(-5000, 3000)
  • To obtain peaks with nearest bi-directional enhancer regions within 5kb upstream and 3kb downstream of shift TSS, set bindingType = "nearest-BiDirectionalPromoters" and bindingRegion = c(-5000, 3000)

startSite  start position of the feature (strand is considered)
endSite  end position of the feature (strand is considered)

nearestBiDirectionalPromoters  nearest enhancer regions from both direction of the peaks (strand is considered). It will report bidirectional enhancer regions if there are enhancer regions in both directions in the given region (defined by bindingRegion). Otherwise, it will report the closest enhancer regions in one direction.

bindingRegion  Annotation range used together with bindingType, which is a vector with two integer values, default to c(-5000, 5000). The first one must be no bigger than 0. And the second one must be no less than 1. For details, see bindingType.

ignore.peak.strand  ignore the peaks strand or not.

...  Not used.
findMotifsInPromoterSeqs

Value

Output is a GRanges object of the annotated peaks.

Author(s)

Jianhong Ou

See Also

See Also as annotatePeakInBatch

Examples

```r
bed <- system.file("extdata", 
                   "wgEncodeUmassDekker5CGm12878PkV2.bed.gz", 
               package="ChIPpeakAnno")
DNAinteractiveData <- toGRanges(gzfile(bed))
library(EnsDb.Hsapiens.v75)
annoData <- toGRanges(EnsDb.Hsapiens.v75, feature="gene")
data("myPeakList")
findEnhancers(myPeakList[500:1000], annoData, DNAinteractiveData)
```

findMotifsInPromoterSeqs

### Description

Find occurrence of input motifs in the promoter regions of the input gene list

### Usage

```r
findMotifsInPromoterSeqs(
    patternFilePath1, 
    patternFilePath2, 
    findPairedMotif = FALSE, 
    BSgenomeName, 
    txdb, 
    geneIDs, 
    upstream = 5000L, 
    downstream = 5000L, 
    name.motif1 = "motif1", 
    name.motif2 = "motif2", 
    max.distance = 100L, 
    min.distance = 1L, 
    motif.orientation = c("both", "motif1UpstreamOfMotif2", "motif2UpstreamOfMotif1"),
    ...) 
```
ignore.strand = FALSE,
format = "fasta",
skip = 0L,
motif1LocForDistance = "end",
motif2LocForDistance = "start",
outfile,
append = FALSE
)

Arguments

patternFilePath1
File path containing a list of known motifs. Required

patternFilePath2
File path containing a motif required to be in the flanking regions of the motif(s) in the first file, i.e., patternFilePath1. Required if findPairedMotif is set to TRUE

findPairedMotif
Find motifs in paired configuration only or not. Default FALSE

BSgenomeName
A BSgenome object. For a list of existing Bsgenomes, please refer use the function available.genomes in BSgenome package. For example, BSgenome.Hsapiens.UCSC.hg38 is for hg38, BSgenome.Hsapiens.UCSC.hg19 is for hg19, BSgenome.Mmusculus.UCSC.mm10 is for mm10, BSgenome.Celegans.UCSC.ce6 is for ce6 BSgenome.Rnorvegicus.UCSC.rn5 is for rn5, BSgenome.Drerio.UCSC.danRer7 is for Zv9, and BSgenome.Dmelanogaster.UCSC.dm3 is for dm3. Required

taxdb

geneIDs
One or more gene entrez IDs. For example the entrez ID for EWSIR is 2130 https://www.genecards.org/cgi-binocardisp.pl?gene=EWSR1 You can use the addGeneIDs function in ChIPpeakAnno to convert other types of Gene IDs to entrez ID

upstream
Number of bases upstream of the TSS to search for the motifs. Default 5000L

downstream
Number of bases downstream of the TSS to search for the motifs. Default 5000L

name.motif1
Name of the motif in inputfilePath2 for labeling the output file column. Default motif1 used only when searching for motifs in paired configuration

name.motif2
Name of the motif in inputfilePath2 for labeling the output file column. Default motif2 used only when searching for motifs in paired configuration

max.distance
maximum required gap between a paired motifs to be included in the output file. Default 100L

min.distance
Minimum required gap between a paired motifs to be included in the output file. Default 1L
motif.orientation

Required relative orientation between paired motifs: both means any orientation, motif1UpstreamOfMotif2 means motif1 needs to be located on the upstream of motif2, and motif2UpstreamOfMotif1 means motif2 needs to be located on the upstream of motif1. Default both

ignore.strand

Specify whether paired motifs should be located on the same strand. Default FALSE

format

The format of the files specified in inputFilePath1 and inputFilePath2. Default fasta

skip

Specify number of lines to skip at the beginning of the input file. Default 0L

motif1LocForDistance

Specify whether to use the start or end of the motif1 location to calculate distance between paired motifs. Only applicable when findPairedMotif is set to TRUE. Default end

motif2LocForDistance

Specify whether to use the start or end of the motif2 location to calculate distance between paired motifs. Only applicable when findPairedMotif is set to TRUE. Default start

outfile

File path to save the search results

append

Specify whether to append the results to the specified output file, i.e., outfile. Default FALSE

Details

This function outputs the motif occurring locations in the promoter regions of input gene list and input motifs. It also can find paired motifs within specified gap threshold

Value

A vector of numeric. It is the background corrected log2-transformed ratios, CPMRatios or Odd-Ratios.

An object of GRanges with metadata "tx_start", "tx_end tx_strand", "tx_id", "tx_name", "Gene ID", and motif specific information such as motif name, motif found, motif strand etc.

Author(s)

Lihua Julie Zhu, Kai Hu

Examples

```
library("BSgenome.Hsapiens.UCSC.hg38")
library("TxDb.Hsapiens.UCSC.hg38.knownGene")

patternFilePath1 = system.file("extdata", "motifIRF4.fa", package="ChIPpeakAnno")
patternFilePath2 = system.file("extdata", "motifAP1.fa", package="ChIPpeakAnno")
pairedMotifs <- findMotifsInPromoterSeqs(patternFilePath1 = patternFilePath1,
                                      patternFilePath2 = patternFilePath2,
                                      ...)```
findPairedMotif = TRUE,
name.motif1 = "IRF4", name.motif2 = "AP1",
BSgenomeName = BSgenome.Hsapiens.UCSC.hg38,
geneIDs = 7486, txdb = TxDb.Hsapiens.UCSC.hg38.knownGene,
outfile = "testPaired.xls")

unPairedMotifs <- findMotifsInPromoterSeqs(patternFilePath1 = patternFilePath1,
  BSgenomeName = BSgenome.Hsapiens.UCSC.hg38,
geneIDs = 7486, txdb = TxDb.Hsapiens.UCSC.hg38.knownGene,
outfile = "testUnPaired.xls")

findOverlappingPeaks

Find the overlapping peaks for two peak ranges.

Description

Find the overlapping peaks for two input peak ranges.

Usage

findOverlappingPeaks(
  Peaks1,
  Peaks2,
  maxgap = -1L,
  minoverlap = 0L,
  multiple = c(TRUE, FALSE),
  NameOfPeaks1 = "TF1",
  NameOfPeaks2 = "TF2",
  select = c("all", "first", "last", "arbitrary"),
  annotate = 0,
  ignore.strand = TRUE,
  connectedPeaks = c("min", "merge",
    ...
  )
)

Arguments

Peaks1    GRanges: See example below.
Peaks2    GRanges: See example below.
maxgap, minoverlap
Used in the internal call to findOverlaps() to detect overlaps. See ?findOverlaps
in the IRanges package for a description of these arguments.
multiple    TRUE or FALSE: TRUE may return multiple overlapping peaks in Peaks2 for
one peak in Peaks1; FALSE will return at most one overlapping peaks in Peaks2
for one peak in Peaks1. This parameter is kept for backward compatibility,
please use select.
findOverlappingPeaks

NameOfPeaks1
Name of the Peaks1, used for generating column name.

NameOfPeaks2
Name of the Peaks2, used for generating column name.

select
all may return multiple overlapping peaks, first will return the first overlapping
peak, last will return the last overlapping peak and arbitrary will return one of
the overlapping peaks.

annotate
Include overlapFeature and shortestDistance in the OverlappingPeaks or not. 1
means yes and 0 means no. Default to 0.

ignore.strand
When set to TRUE, the strand information is ignored in the overlap calculations.

connectedPeaks
If multiple peaks involved in overlapping in several groups, set it to "merge"
will count it as only 1, while set it to "min" will count it as the minimal involved
peaks in any concerned groups

... Objects of GRanges: See also findOverlapsOfPeaks.

Details
The new function findOverlapsOfPeaks is recommended.
Efficiently perform overlap queries with an interval tree implemented in IRanges.

Value
OverlappingPeaks
a data frame consists of input peaks information with added information: overlapFeature (upstream: peak1 resides upstream of the peak2; downstream: peak1 resides downstream of the peak2; inside: peak1 resides inside the peak2 entirely; overlapStart: peak1 overlaps with the start of the peak2; overlapEnd: peak1 overlaps with the end of the peak2; includeFeature: peak1 include the peak2 entirely) and shortestDistance (shortest distance between the overlapping peaks)

MergedPeaks
GRanges contains merged overlapping peaks

Author(s)
Lihua Julie Zhu

References

See Also
findOverlapsOfPeaks, annotatePeakInBatch, makeVennDiagram
Examples
if (interactive())
{
peaks1 =
GRanges(seqnames=c(6,6,6,6,5),
IRanges(start=c(1543200,1557200,1563000,1569800,167889600),
   end=c(1555199,1560599,1565199,1573799,167893599),
   names=c("p1","p2","p3","p4","p5"),
   strand=as.integer(1))
peaks2 =
GRanges(seqnames=c(6,6,6,6,5),
IRanges(start=c(1549800,1554400,1565000,1569400,167888600),
   end=c(1550599,1560799,1565399,1571199,167888999),
   names=c("f1","f2","f3","f4","f5"),
   strand=as.integer(1))
t1 =findOverlappingPeaks(peaks1, peaks2, maxgap=1000,
   NameOfPeaks1="TF1", NameOfPeaks2="TF2", select="all", annotate=1)
r = t1$OverlappingPeaks
pie(table(r$overlapFeature))
as.data.frame(t1$MergedPeaks)
}

findOverlapsOfPeaks

Find the overlapped peaks among two or more set of peaks.

Description
Find the overlapping peaks for two or more (less than five) set of peak ranges.

Usage
findOverlapsOfPeaks(
   ...
   maxgap = -1L,
   minoverlap = 0L,
   ignore.strand = TRUE,
   connectedPeaks = c("keepAll", "min", "merge")
)

Arguments

   ... Objects of GRanges: See example below.
maxgap, minoverlap

   Used in the internal call to findOverlaps() to detect overlaps. See ?findOverlaps in the IRanges package for a description of these arguments. If 0 < minoverlap < 1, the function will find overlaps by percentage covered of interval and the filter condition will be set to max covered percentage of overlapping peaks.
findOverlapsOfPeaks

ignore.strand When set to TRUE, the strand information is ignored in the overlap calculations.

connectedPeaks If multiple peaks are involved in any group of connected/overlapping peaks in any input peak list, set it to "merge" will add 1 to the overlapping counts, while set it to "min" will add the minimal involved peaks in each group of connected/overlapped peaks to the overlapping counts. Set it to "keepAll" will add the number of involved peaks for each peak list to the corresponding overlapping counts. In addition, it will output counts as if connectedPeaks were set to "min". For examples (https://support.bioconductor.org/p/133486/#133603), if 5 peaks in group1 overlap with 2 peaks in group 2, setting connectedPeaks to "merge" will add 1 to the overlapping counts; setting it to "keepAll" will add 5 peaks to count.group1, 2 to count.group2, and 2 to counts; setting it to "min" will add 2 to the overlapping counts.

Details

Efficiently perform overlap queries with an interval tree implemented with GRanges.

Value

return value is An object of overlappingPeaks.

venn_cnt an object of VennCounts
peaklist a list consists of all overlapping peaks or unique peaks
uniquePeaks an object of GRanges consists of all unique peaks
mergedPeaks an object of GRanges consists of all merged overlapping peaks
peaksInMergedPeaks an object of GRanges consists of all peaks in each samples involved in the overlapping peaks
overlappingPeaks a list of data frame consists of the annotation of all the overlapped peaks
all.peaks a list of GRanges object which contain the input peaks with formatted rownames.

Author(s)

Jianhong Ou

References


genomicElementDistribution

Genomic Element distribution

Description

Plot pie chart for genomic element distribution

Usage

```r
genomicElementDistribution(
  peaks,
  TxDb,
  seqlev,
  nucleotideLevel = FALSE,
  ignore.strand = TRUE,
  promoterRegion = c(upstream = 2000, downstream = 100),
  geneDownstream = c(upstream = 0, downstream = 1000),
  labels = list(geneLevel = c(promoter = "Promoter", geneDownstream = "Downstream",
    geneBody = "Gene body", distalIntergenic = "Distal Intergenic"),
    ExonIntron = c(exon = "Exon", intron = "Intron", intergenic = "Intergenic"),
    Exons = c(utr5 = "5' UTR", utr3 = "3' UTR", CDS = "CDS", otherExon = "Other exon"),
    group = c(geneLevel =
```
"Transcript Level", promoterLevel = "Promoter Level", Exons = "Exon level",
ExonIntron = "Exon/Intron/Intergenic"),
labelColors = c(promoter = "#E1F114", geneBody = "#9EFF00", geneDownstream = "#57CB1B",
distalIntergenic = "#066A4B", exon = "#6600FF", intron = "#8F00FF", intergenic =
"#DA00FF", utr5 = "#00FFDB", utr3 = "#00DFFF", CDS = "#00A0FF", otherExon =
"#006FFF"),
plot = TRUE,
keepExonsInGenesOnly = TRUE,
promoterLevel
)

Arguments

peaks peak list, GRanges object or a GRangesList.

TxDb an object of TxDb

seqlev sequence level should be involved. Default is all the sequence levels in intersect
of peaks and TxDb.

nucleotideLevel Logical. Choose between peak centric and nucleotide centric view. Default=FALSE

ignore.strand logical. Whether the strand of the input ranges should be ignored or not. De-
default=TRUE

promoterRegion numeric. The upstream and downstream of genes to define promoter region.

geneDownstream numeric. The upstream and downstream of genes to define gene downstream
region.

labels list. A list for labels for the genomic elements.

labelColors named character vector. The colors for each labels.

plot logic. Plot the pie chart for the genomic elements or not.

keepExonsInGenesOnly logic. Keep the exons within annotated gene only.

promoterLevel list. The breaks, labels, and colors for divided range of promoters. The breaks
must be from 5’ -> 3’ and the percentage will use the fixed precedence 3’ -> 5’

Details

The distribution will be calculated by geneLevel, ExonIntron, and Exons The geneLevel will be
categorized as promoter region, gene body, gene downstream and distal intergenic region. The
ExonIntron will be categorized as exon, intron and intergenic. The Exons will be categorized as 5’
UTR, 3’UTR and CDS. The precedence will follow the order of labels definition. For example, for
ExonIntron, if a peak overlap with both exon and intron, and exon is specified before intron, then
only exon will be incremented for the same example.

Value

Invisible list of data for plot.
```
Examples

if (interactive() || Sys.getenv("USER")=='jianhongou'){
  data(myPeakList)
  if(require(TxDb.Hsapiens.UCSC.hg19.knownGene)){
    seqinfo(myPeakList) <-
    seqinfo(TxDb.Hsapiens.UCSC.hg19.knownGene)[seqlevels(myPeakList)]
    myPeakList <- GenomicRanges::trim(myPeakList)
    myPeakList <- myPeakList[width(myPeakList)>0]
    genomicElementDistribution(myPeakList,
      TxDb.Hsapiens.UCSC.hg19.knownGene)
    genomicElementDistribution(myPeakList,
      TxDb.Hsapiens.UCSC.hg19.knownGene,
      nucleotideLevel = TRUE)
    genomicElementDistribution(myPeakList,
      TxDb.Hsapiens.UCSC.hg19.knownGene,
      promoterLevel=list(
        "from 5' -> 3'" , fixed precedence 3' -> 5'
        breaks = c(-2000, -1000, -500, 0, 100),
        labels = c("upstream 1-2Kb", "upstream 0.5-1Kb",
                  "upstream <500b", "TSS - 100b"),
        colors = c("#FFE5CC", "#FFCA99",
                  "#FFAD65", "#FF8E32")))
  }
}
```

---

genomicElementUpSetR  Genomic Element data for upset plot

---

### Description

Prepare data for upset plot for genomic element distribution

### Usage

```r
genomicElementUpSetR(
  peaks,
  TxDB,
  seqlev,
  ignore.strand = TRUE,
  breaks = list(distal_upstream = c(-1e+05, -10000, -1, 1), proximal_upstream = c(-10000, -5000, -1, 1), distal_promoter = c(-5000, -2000, -1, 1), proximal_promoter = c(-2000, 200, -1, 0), '5'UTR' = fiveUTRsByTranscript, '3'UTR' = threeUTRsByTranscript, CDS = cds, exon = exons, intron = intronsByTranscript, gene_body = genes, immediate_downstream = c(0, 2000, 1, 1), proximal_downstream = c(2000, 5000, 1, 1),
  distal_downstream = c(5000, 1e+05, 1, 1))
)```
**getAllPeakSequence**

**Arguments**

- **peaks**: peak list, GRanges object or a GRangesList.
- **TxDb**: an object of TxDb
- **seqlev**: sequence level should be involved. Default is all the sequence levels in intersect of peaks and TxDb.
- **ignore.strand**: logical. Whether the strand of the input ranges should be ignored or not. Default=TRUE
- **breaks**: list. A list for labels and sets for the genomic elements. The element could be an S4 method for signature 'TxDb' or a numeric vector with length of 4. The three numbers are c(upstream point, downstream point, promoter (-1) or downstream (1), remove gene body or not (1: remove, 0: keep)).

**Details**

The data will be calculated by for each breaks. No precedence will be considered.

**Value**

list of data for plot.

**Examples**

```r
if (interactive() || Sys.getenv("USER")="$jianhongou"){
  data(myPeakList)
  if(require(TxDb.Hsapiens.UCSC.hg19.knownGene)){
    seqinfo(myPeakList) <-
    seqinfo(TxDb.Hsapiens.UCSC.hg19.knownGene)[seqlevels(myPeakList)]
    myPeakList <- GenomicRanges::trim(myPeakList)
    myPeakList <- myPeakList[width(myPeakList)>0]
    x <- genomicElementUpSetR(myPeakList,
      TxDb.Hsapiens.UCSC.hg19.knownGene)
    library(UpSetR)
    upset(x$plotData, nsets=13, nintersects=NA)
  }
}
```

---

**getAllPeakSequence**: Obtain genomic sequences around the peaks

**Description**

Obtain genomic sequences around the peaks leveraging the BSgenome and biomaRt package
getAllPeakSequence

Usage

generalPeakSequence(
  myPeakList,
  upstream = 200L,
  downstream = upstream,
  genome,
  AnnotationData
)

Arguments

myPeakList An object of GRanges: See example below
upstream upstream offset from the peak start, e.g., 200
downstream downstream offset from the peak end, e.g., 200
gene BSgenome object or mart object. Please refer to available.genomes in BSgenome package and useMart in bioMaRt package for details
AnnotationData GRanges object with annotation information.

Value

GRanges with slot start holding the start position of the peak, slot end holding the end position of the peak, slot rownames holding the id of the peak and slot seqnames holding the chromosome where the peak is located. In addition, the following variables are included:

upstream upstream offset from the peak start
downstream downstream offset from the peak end
sequence the sequence obtained

Author(s)

Lihua Julie Zhu, Jianhong Ou

References


Examples

#### use Annotation data from BSgenome
peaks <- GRanges(seqnames=c("NC_008253", "NC_010468"),
  IRanges(start=c(100, 500), end=c(300, 600),
    names=c("peak1", "peak2"))
library(BSgenome.Ecoli.NCBI.20080805)
se <- getAllPeakSequence(peaks, upstream=20, downstream=20, genome=Ecoli)
write2FASTA(seq, file="test.fa")
getAnnotation  

Obtain the TSS, exon or miRNA annotation for the specified species

Description

Obtain the TSS, exon or miRNA annotation for the specified species using the biomaRt package.

Usage

getAnnotation(
  mart,
  featureType = c("TSS", "miRNA", "Exon", "5utr", "3utr", "ExonPlusUtr", "transcript")
)

Arguments

mart      A mart object, see useMart of biomaRt package for details.
featureType TSS, miRNA, Exon, 5'UTR, 3'UTR, transcript or Exon plus UTR. The default is TSS.

Value

GRanges with slot start holding the start position of the feature, slot end holding the end position of the feature, slot names holding the id of the feature, slot space holding the chromosome location where the feature is located. In addition, the following variables are included.

list("strand")  1 for positive strand and -1 for negative strand where the feature is located
list("description")  description of the feature such as gene

Note

For featureType of TSS, start is the transcription start site if strand is 1 (plus strand), otherwise, end is the transcription start site.

Note that the version of the annotation db must match with the genome used for mapping because the coordinates may differ for different genome releases. For example, if you are using Mus_musculus.v103 for mapping, you’d best also use EnsDb.Mmusculus.v103 for annotation. See Examples for more info.

Author(s)

Lihua Julie Zhu, Jianhong Ou, Kai Hu

References

getEnrichedGO

Obtain enriched gene ontology (GO) terms that near the peaks

description
Obtain enriched gene ontology (GO) terms based on the features near the enriched peaks using GO.db package and GO gene mapping package such as org.Hs.db to obtain the GO annotation and using hypergeometric test (phyper) and multtest package for adjusting p-values

Usage
getEnrichedGO(
  annotatedPeak,
  orgAnn,
  feature_id_type = "ensembl_gene_id",
  maxP = 0.01,
**getEnrichedGO**

```r
minGOTerm = 10,
multiAdjMethod = NULL,
condense = FALSE,
removeAncestorByPval = NULL,
keepByLevel = NULL,
subGroupComparison = NULL
```

**Arguments**

- `annotatedPeak` A GRanges object or a vector of feature IDs
- `orgAnn` Organism annotation package such as `org.Hs.eg.db` for human and `org.Mm.eg.db` for mouse, `org.Dm.eg.db` for fly, `org.Rn.eg.db` for rat, `org.Sc.eg.db` for yeast and `org.Dr.eg.db` for zebrafish
- `feature_id_type` The feature type in annotatedPeak such as `ensembl_gene_id`, `refseq_id`, `gene_symbol` or `entrez_id`
- `maxP` The maximum p-value to be considered to be significant
- `minGOTerm` The minimum count in a genome for a GO term to be included
- `multiAdjMethod` The multiple testing procedures, for details, see `mt.rawp2adjp` in `multtest` package
- `condense` Condense the results or not.
- `removeAncestorByPval` Remove ancestor by p-value. P-value is calculated by fisher exact test. If gene number in all of the children is significant greater than it in parent term, the parent term will be removed from the list.
- `keepByLevel` If the shortest path from the go term to 'all' is greater than the given level, the term will be removed.
- `subGroupComparison` A logical vector to split the peaks into two groups. The enrichment analysis will compare the over-present GO terms in TRUE group and FALSE group separately. The analysis will split into two steps: 1. enrichment analysis for TRUE group by hypergeometric test; 2. enrichment analysis for TRUE over FALSE group by Fisher’s Exact test for the enriched GO terms. To keep the output same format, if you want to compare FALSE vs TRUE, please repeat the analysis by inverting the parameter. Default is NULL.

**Value**

A list with 3 elements

- `list("bp")` enriched biological process with the following 9 variables
  - `go.id`: GO biological process id
  - `go.term`: GO biological process term
  - `go.Definition`: GO biological process description
  - Ontology: Ontology branch, i.e. BP for biological process
count.InDataset: count of this GO term in this dataset
count.InGenome: count of this GO term in the genome
pvalue: pvalue from the hypergeometric test
totaltermInDataset: count of all GO terms in this dataset
totaltermInGenome: count of all GO terms in the genome

list("mf")
enriched molecular function with the following 9 variables
go.id:GO molecular function id
go.term:GO molecular function term
go.Definition:GO molecular function description
Ontology: Ontology branch, i.e. MF for molecular function
count.InDataset: count of this GO term in this dataset
count.InGenome: count of this GO term in the genome
pvalue: pvalue from the hypergeometric test
totaltermInDataset: count of all GO terms in this dataset
totaltermInGenome: count of all GO terms in the genome

list("cc")
enriched cellular component the following 9 variables
go.id:GO cellular component id
go.term:GO cellular component term
go.Definition:GO cellular component description
Ontology: Ontology type, i.e. CC for cellular component
count.InDataset: count of this GO term in this dataset
count.InGenome: count of this GO term in the genome
pvalue: pvalue from the hypergeometric test
totaltermInDataset: count of all GO terms in this dataset
totaltermInGenome: count of all GO terms in the genome

Author(s)
Lihua Julie Zhu. Jianhong Ou for subGroupComparison

References

See Also
phyper, hyperGtest

Examples
data(enrichedGO)
enrichedGO$mf[1:10,]
enrichedGO$bp[1:10,]
enrichedGO$cc
eval(
  if (interactive()) {
    
  
} else {
  
})
getEnrichedPATH

Obtain enriched PATH that near the peaks

Description
Obtain enriched PATH that are near the peaks using path package such as reactome.db and pathway mapping package such as org.Hs.db.eg to obtain the path annotation and using hypergeometric test (phyper) and multtest package for adjusting p-values

Usage
getEnrichedPATH(
  annotatedPeak,
  orgAnn,
  pathAnn,
  feature_id_type = "ensembl_gene_id",
  maxP = 0.01,
  minPATHterm = 10,
  multiAdjMethod = NULL,
  subGroupComparison = NULL
)

Arguments
annotatedPeak GRanges such as data(annotatedPeak) or a vector of feature IDs
orgAnn organism annotation package such as org.Hs.db.eg for human and org.Mm.db.eg for mouse, org.Dm.db.eg for fly, org.Rn.db.eg for rat, org.Sc.db.eg for yeast and org.Dr.db.eg for zebrafish
pathAnn pathway annotation package such as KEGG.db (deprecated), reactome.db, KEG-GREST
feature_id_type the feature type in annotatedPeakRanges such as ensembl_gene_id, refseq_id, gene_symbol or entrez_id
getEnrichedPATH

maxP maximum p-value to be considered to be significant
minPATHterm minimum count in a genome for a path to be included
multiAdjMethod multiple testing procedures, for details, see mt.rawp2adjp in multtest package
subGroupComparison A logical vector to split the peaks into two groups. The enrichment analysis will compare the over-present GO terms in TRUE group and FALSE group separately. The analysis will split into two steps: 1. enrichment analysis for TRUE group by hypergeometric test; 2. enrichment analysis for TRUE over FALSE group by Fisher’s Exact test for the enriched GO terms. To keep the output same format, if you want to compare FALSE vs TRUE, please repeat the analysis by inverting the parameter. Default is NULL.

Value
A dataframe of enriched path with the following variables.

path.id KEgg PATH ID
EntrezID Entrez ID
count.InDataset count of this PATH in this dataset
count.InGenome count of this PATH in the genome
pvalue pvalue from the hypergeometric test
totaltermInDataset count of all PATH in this dataset
totaltermInGenome count of all PATH in the genome
PATH PATH name

Author(s)
Jianhong Ou, Kai Hu

References

See Also
phyper, hyperGtest

Examples
if (interactive()||Sys.getenv("USER")=="jianhongou") {
data(annotatedPeak)
library(org.Hs.eg.db)
library(reactome.db)
enriched.PATH = getEnrichedPATH(annotatedPeak, orgAnn="org.Hs.eg.db"),
getGeneSeq

Get gene sequence using the biomaRt package

Description
Get gene sequence using the biomaRt package

Usage
getGeneSeq(LocationParameters, mart)

Arguments
LocationParameters
c(ensembl_gene_id, distance from the peak to the transcription start site of the
gene with the above ensemblID, upstream offset from the peak, downstream
offset from the peak, Gene Start, Gene End)
mart see useMart of bioMaRt package for details

Value
a list with the following items

feature_id ensemble gene ID
distancetoFeature distance from the peak to the transcription start site of the gene with the above
ensembl gene ID
upstream upstream offset from the peakStart
downstream downstream offset from the peakEnd
seq sequence obtained around the peak with above upstream and downstream offset

Note
internal function not intended to be called directly by users
getGO

Author(s)
Lihua Julie Zhu

Examples

if (interactive())
{
  mart <- useMart(biomart="ensembl", dataset="drerio_gene_ensembl")
  getGeneSeq(LocationParameters =c("ENSDARG00000054562",400, 750, 750,40454140,40454935), mart)

  LocationParameters =c("ENSDARG00000054562",752, 750, 750,40454140,40454935)
  getGeneSeq(LocationParameters, mart)

  LocationParameters =c("ENSDARG00000054562",750, 750, 750,40454140,40454935)
  getGeneSeq(LocationParameters, mart)

  LocationParameters =c("ENSDARG00000054562",-2, 750, 750,40454140,40454935)
  getGeneSeq(LocationParameters, mart)

  LocationParameters =c("ENSDARG00000054562",0, 750, 750,40454140,40454935)
  getGeneSeq(LocationParameters, mart)

  LocationParameters =c("ENSDARG00000054562",2, 750, 750,40454140,40454935)
  getGeneSeq(LocationParameters, mart)

  LocationParameters =c("ENSDARG00000054562",1000, 750, 750,40454140,40454935)
  getGeneSeq(LocationParameters, mart)
}

getGO

Obtain gene ontology (GO) terms for given genes

Description
Obtain gene ontology (GO) terms useing GO gene mapping package such as org.Hs.db.eg to obtain the GO annotation.

Usage
getGO(all.genes, orgAnn = "org.Hs.eg.db", writeTo, ID_type = "gene_symbol")
getUniqueGOidCount

Arguments

all.genes A character vector of feature IDs
orgAnn Organism annotation package such as org.Hs.eg.db for human and org.Mm.eg.db for mouse, org.Dm.eg.db for fly, org.Rn.eg.db for rat, org.Sc.eg.db for yeast and org.Dr.eg.db for zebrafish
writeTo File path for output table
ID_type The feature type in annotatedPeak such as ensembl_gene_id, refseq_id, gene_symbol

Value
An invisible table with genes and GO terms.

Author(s)
Lihua Julie Zhu

See Also
getEnrichedGO

Examples

if (interactive()) {
  data(annotatedPeak)
  library(org.Hs.eg.db)
  getGO(annotatedPeak[1:6]$feature,
       orgAnn="org.Hs.eg.db",
       ID_type="ensembl_gene_id")
}

getUniqueGOidCount

get the count for each unique GO ID

Description
get the count for each unique GO ID

Usage
getUniqueGOidCount(goList)

Arguments

goList a set of GO terms as character vector
Value

a list with 2 variables

 GOterm a vector of GO terms as character vector
 GOcount counts corresponding to the above GOterm as numeric vector

Note

internal function not intended to be called directly by users

Author(s)

Lihua Julie Zhu

See Also

getEnrichedGO

Examples

       "GO:0000122", "GO:0000122", "GO:0000075", "GO:0000082", "GO:000012")

getUniqueGOidCount(goList)
getVennCounts

Arguments

... Objects of GRanges. See example below.

maxgap, minoverlap

Used in the internal call to findOverlaps() to detect overlaps. See ?findOverlaps in the IRanges package for a description of these arguments.

by region, feature or base, default region. feature means using feature field in the GRanges for calculating overlap, region means using chromosome range for calculating overlap, and base means using calculating overlap in nucleotide level.

ignore.strand

When set to TRUE, the strand information is ignored in the overlap calculations.

connectedPeaks

If multiple peaks involved in overlapping in several groups, set it to "merge" will count it as only 1, while set it to "min" will count it as the minimal involved peaks in any concerned groups

Value

vennCounts vennCounts objects containing counts for Venn Diagram generation, see details in limma package vennCounts

Author(s)

Jianhong Ou

See Also

makeVennDiagram, findOverlappingPeaks

Examples

if(interactive() || Sys.getenv("USER")=='jianhongou'){
  peaks1 = GRanges(seqnames=c("1", "2", "3"),
                  IRanges(start = c(967654, 2010897, 2496704),
                          end = c(967754, 2010997, 2496804),
                          names = c("Site1", "Site2", "Site3")),
                  strand=as.integer(1),
                  feature=c("a","b", "c"))
  peaks2 =
        GRanges(seqnames= c("1", "2", "3", "1", "2"),
        IRanges(start=c(967659, 2010898, 2496700, 3075866, 3123260),
                end=c(967869, 2011108, 2496920, 3076166, 3123470),
                names = c("t1", "t2", "t3", "t4", "t5")),
                strand = c(1L, 1L, -1L,-1L,1L),
                feature=c("a","c","d","e", "a"))
  getVennCounts(peaks1,peaks2)
  getVennCounts(peaks1,peaks2, by="feature")
  getVennCounts(peaks1, peaks2, by="base")
}
**HOT.spots**  
*High Occupancy of Transcription Related Factors regions*

**Description**
High Occupancy of Transcription Related Factors regions of human (hg19)

**Usage**

```r
HOT.spots
```

**Format**
An object of GRangesList

**Details**
How to generated the data:
```r
temp <- tempfile()
url <- "http://metatracks.encodenets.gersteinlab.org"
download.file(file.path(url, "HOT_All_merged.tar.gz"), temp)
temp2 <- tempfile()
download.file(file.path(url, "HOT_intergenic_All_merged.tar.gz"), temp2)
untar(temp, exdir=dirname(temp))
untar(temp2, exdir=dirname(temp))
f <- dir(dirname(temp), "bed$")
HOT.spots <- sapply(file.path(dirname(temp), f), toGRanges, format="BED")
names(HOT.spots) <- gsub("_merged.bed", "", f)
HOT.spots <- sapply(HOT.spots, unname)
HOT.spots <- GRangesList(HOT.spots)
save(list="HOT.spots",
     file="data/HOT.spots.rda",
     compress="xz", compression_level=9)
```

**Source**
http://metatracks.encodenets.gersteinlab.org/
References


Examples

data(HOT.spots)
elementNROWs(HOT.spots)

<table>
<thead>
<tr>
<th>hyperGtest</th>
<th>hypergeometric test</th>
</tr>
</thead>
</table>

Description

hypergeometric test with lower.tail = FALSE used by getEnrichedGO

Usage

hyperGtest(alltermcount, thistermcount, totaltermInGenome, totaltermInPeakList)

Arguments

- alltermcount: a list with two variables: GOterm and GOcount which is GO terms and corresponding counts in the whole genome
- thistermcount: a list with two variables: GOterm and GOcount which is GO terms and corresponding counts in the peak list
- totaltermInGenome: number of total GO terms in the whole genome
- totaltermInPeakList: number of total GO terms in the peak list

Details

see phyper for details

Value

a list with 6 variables

- thisterm: GO term
- thistermcount: count of this GO term in the peak list
- thistermtotal: count of this GO term in the whole genome
pvalue        pvalue of the hypergeometric test
 totaltermInPeakList
    number of total GO terms in the peak list
 totaltermInGenome
    number of total GO terms in the whole genome

**Note**
internal function not intended to be used directly by users

**Author(s)**
Lihua Julie ZHu

**References**

**See Also**
phyper, getEnrichedGO

**Examples**
```r
            "GO:0000122", "GO:0000075", "GO:0000082", "GO:0000082",
            "GO:0000122", "GO:0000122", "GO:0000122", "GO:0000122",
            "GO:0000075", "GO:0000082", "GO:000012")

alltermcount = list(GOterm=c("GO:0000075", "GO:0000082", "GO:000012",
                              "GO:0000122"),
                    GOcount=c(100, 200, 10, 10))

thistermcount = getUniqueGOidCount(goList)
totaltermInPeakList = 15
totaltermInGenome = 1000
hyperGtest(alltermcount, thistermcount, totaltermInGenome, totaltermInPeakList)
```

---

**IDRfilter**

*Filter peaks by IDR (irreproducible discovery rate)*

**Description**
Using IDR to assess the consistency of replicate experiments and obtain a high-confidence single set of peaks
IDRfilter

Usage

IDRfilter(
  peaksA,
  peaksB,
  bamfileA,
  bamfileB,
  maxgap = -1L,
  minoverlap = 0L,
  singleEnd = TRUE,
  IDRcutoff = 0.01,
  ...
)

Arguments

  peaksA, peaksB  peaklist, GRanges object.
  bamfileA, bamfileB  file path of bam files.
  maxgap, minoverlap  Used in the internal call to findOverlaps() to detect overlaps. See ?findOverlaps in the IRanges package for a description of these arguments.
  singleEnd  (Default TRUE) A logical indicating if reads are single or paired-end.
  IDRcutoff  If the IDR no less than IDRcutoff, the peak will be removed.
  ...  Not used.

Value

  An object GRanges

Author(s)

  Jianhong Ou

References


Examples

if(interactive()){
  path <- system.file("extdata", "reads", package="MMDiffBamSubset")
  if(file.exists(path)){
    bamfileA <- file.path(path, "reads", "WT_2.bam")
    bamfileB <- file.path(path, "reads", "Resc_2.bam")
    WT.AB2.Peaks <- file.path(path, "peaks", "WT_2_Macs_peaks.xls")
    Resc.AB2.Peaks <- file.path(path, "peaks",
                              "Resc_2_Macs_peaks.xls")
  }
}
peaksA=toGRanges(WT.AB2.Peaks, format="MACS")
peaksB=toGRanges(Resc.AB2.Peaks, format="MACS")
library(idr)
library(DelayedArray)
IDRfilter(peaksA, peaksB,
        bamfileA, bamfileB)
}
)

makeVennDiagram

Make Venn Diagram from a list of peaks

Description

Make Venn Diagram from two or more peak ranges, Also calculate p-value to determine whether those peaks overlap significantly.

Usage

makeVennDiagram(
  Peaks,
  NameOfPeaks,  
  maxgap = -1L,  
  minoverlap = 0L,
  totalTest,
  by = c("region", "feature", "base"),
  ignore.strand = TRUE,
  connectedPeaks = c("min", "merge", "keepAll", "keepFirstListConsistent"),
  method = c("hyperG", "permutation"),
  TxDb,
  plot = TRUE,
  ...
)

Arguments

Peaks A list of peaks in GRanges format: See example below.

NameOfPeaks Character vector to specify the name of Peaks, e.g., c("TF1", "TF2"). This will be used as label in the Venn Diagram.

maxgap, minoverlap Used in the internal call to findOverlaps() to detect overlaps. See ?findOverlaps in the IRanges package for a description of these arguments.

totalTest Numeric value to specify the total number of tests performed to obtain the list of peaks. It should be much larger than the number of peaks in the largest peak set.
makeVennDiagram

by "region", "feature" or "base", default = "region". "feature" means using feature field in the GRanges for calculating overlap, "region" means using chromosome range for calculating overlap, and "base" means calculating overlap in nucleotide level.

ignore.strand Logical: when set to TRUE, the strand information is ignored in the overlap calculations.

connectedPeaks If multiple peaks involved in overlapping in several groups, set it to "merge" will count it as only 1, while set it to "min" will count it as the minimal involved peaks in any connected peak group. "keepAll" will show all the orginal counts for each list while the final counts will be same as "min". "keepFirstListConsistent" will keep the counts consistent with first list.

method method to be used for p value calculation. hyperG means hypergeometric test and permutation means peakPermTest.

TxDb An object of TxDb.

plot logical. If TRUE (default), a venn diagram is plotted.

... Additional arguments to be passed to venn.diagram.

Details

For customized graph options, please see venn.diagram in VennDiagram package.

Value

A p.value is calculated by hypergeometric test or permutation test to determine whether the overlaps of peaks or features are significant.

Author(s)

Lihua Julie Zhu, Jianhong Ou

See Also

findOverlapsOfPeaks, venn.diagram, peakPermTest

Examples

if (interactive()){
  peaks1 <- GRanges(seqnames=c("1", "2", "3"),
    IRanges(start=c(967654, 2010897, 2496704),
      end=c(967754, 2010997, 2496804),
      names=c("Site1", "Site2", "Site3")),
    strand="+",
    feature=c("a", "b", "f"))
  peaks2 = GRanges(seqnames=c("1", "2", "3", "1", "2"),
    IRanges(start = c(967659, 2010898, 2496700,
      3075860, 3123260),
    end = c(967869, 2011108, 2496920,
      3076166, 3123470),
    names = c("t1", "t2", "t3", "t4", "t5")),
  ...
mergePlusMinusPeaks

Merge peaks from plus strand and minus strand

Description

Merge peaks from plus strand and minus strand within certain distance apart, and output merged peaks as bed format.

Usage

mergePlusMinusPeaks(
  peaks.file,
  columns = c("name", "chromosome", "start", "end", "strand", "count", "count", "count", "count"),
  sep = "\t",
  header = TRUE,
  distance.threshold = 100,
  plus.strand.start.gt.minus.strand.end = TRUE,
  output.bedfile
)

Arguments

peaks.file Specify the peak file. The peak file should contain peaks from both plus and minus strand

columns Specify the column names in the peak file
mergePlusMinusPeaks

- **sep**: Specify column delimiter, default tab-delimited
- **header**: Specify whether the file has a header row, default TRUE
- **distance.threshold**: Specify the maximum gap allowed between the plus stranded and the negative stranded peak
- **plus.strand.start.gt.minus.strand.end**: Specify whether plus strand peak start greater than the paired negative strand peak end. Default to TRUE
- **output.bedfile**: Specify the bed output file name

**Value**

output the merged peaks in bed file and a data frame of the bed format

**Author(s)**

Lihua Julie Zhu

**References**


**See Also**

annotatePeakInBatch, findOverlappingPeaks, makeVennDiagram

**Examples**

```r
if (interactive()) {
  data(myPeakList)
  data(TSS.human.NCBI36)
  library(matrixStats)
  peaks <- system.file("extdata", "guide-seq-peaks.txt", package = "ChIPpeakAnno")
  merged.bed <- mergePlusMinusPeaks(peaks.file = peaks,
                                    columns=c("name", "chromosome",
                                              "start", "end", "strand",
                                              "count", "count"),
                                    sep = "\t", header = TRUE,
                                    distance.threshold = 100,
                                    plus.strand.start.gt.minus.strand.end = TRUE,
                                    output.bedfile = "T2test100bp.bed")
}
```
Description

Bar plot for distance to features

Usage

metagenePlot(
  peaks, AnnotationData, 
  PeakLocForDistance = c("middle", "start", "end"), 
  FeatureLocForDistance = c("TSS", "middle", "geneEnd"), 
  upstream = 1e+05, 
  downstream = 1e+05
)

Arguments

peaks peak list, GRanges object or a GRangesList.
AnnotationData A GRanges object or a TxDb object.
PeakLocForDistance Specify the location of peak for calculating distance, i.e., middle means using middle of the peak to calculate distance to feature, start means using start of the peak to calculate the distance to feature. To be compatible with previous version, by default using start.
FeatureLocForDistance Specify the location of feature for calculating distance, i.e., middle means using middle of the feature to calculate distance of peak to feature, TSS means using start of feature when feature is on plus strand and using end of feature when feature is on minus strand, geneEnd means using end of feature when feature is on plus strand and using start of feature when feature is on minus strand.
upstream, downstream numeric(1). Upstream or downstream region of features to plot.

Details

the bar heatmap is indicates the peaks around features.

Examples

path <- system.file("extdata", package="ChIPpeakAnno")
files <- dir(path, "broadPeak")
peaks <- sapply(file.path(path, files), toGRanges, format="broadPeak")
peaks <- GRangesList(peaks)
names(peaks) <- sub(".broadPeak", ",", basename(names(peaks)))
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
metagenePlot(peaks, TxDb.Hsapiens.UCSC.hg19.knownGene)

myPeakList
An example GRanges object representing a ChIP-seq peak dataset

Description

the putative STAT1-binding regions identified in un-stimulated cells using ChIP-seq technology
(Robertson et al., 2007)

Usage

myPeakList

Format

GRanges with slot rownames containing the ID of peak as character, slot start containing the start
position of the peak, slot end containing the end position of the peak and seqnames containing the
chromosome where the peak is located.

Source

STAT1 DNA association using chromatin immunoprecipitation and massively parallel sequencing.
Nat Methods 4:651-7

Examples

data(myPeakList)
slotNames(myPeakList)

oligoFrequency
get the oligonucleotide frequency

Description

Prepare the oligonucleotide frequency for given Markov order.

Usage

oligoFrequency(sequence, MarkovOrder = 3L)

Arguments

sequence The sequences packaged in DNAStringSet, DNAString object or output of func-
tion getAllPeakSequence.
MarkovOrder Markov order.
Value
A numeric vector.

Author(s)
Jianhong Ou

See Also
See Also as oligoSummary

Examples

library(seqinr)
library(Biostrings)
oligoFrequency(DNAString("AATTCGACGTACAGATGACTAGACT"))

oligoSummary
Output a summary of consensus in the peaks

Description
Calculate the z-scores of all combinations of oligonucleotide in a given length by Markov chain.

Usage

oligoSummary(
  sequence,
  oligoLength = 6L,
  freqs = NULL,
  MarkovOrder = 3L,
  quickMotif = FALSE,
  revcomp = FALSE,
  maxsize = 1e+05
)

Arguments

sequence The sequences packaged in DNAStringSet, DNAString object or output of function getAllPeakSequence.
oligoLength The length of oligonucleotide.
freqs Output of function frequency.
MarkovOrder The order of Markov chain.
quickMotif Generate the motif by z-score of not.
revcomp Consider both the given strand and the reverse complement strand when searching for motifs in a complementable alphabet (ie DNA). Default, FALSE.
maxsize Maximum allowed dataset size (in length of sequences).
peakPermTest

Value

A list is returned.

- **zscore**: A numeric vector. The z-scores of each oligonucleotide.
- **counts**: A numeric vector. The counts number of each oligonucleotide.
- **motifs**: A list of motif matrix.

Author(s)

Jianhong Ou

References


See Also

See Also as `frequency`

Examples

```r
if(interactive() || Sys.getenv("USER")=="jianhongou"){
  data(annotatedPeak)
  library(BSgenome.Hsapiens.UCSC.hg19)
  library(seqinr)
  seq <- getAllPeakSequence(annotatedPeak[1:100],
    upstream=20,
    downstream=20,
    genome=Hsapiens)
  oligoSummary(seq)
}
```

<table>
<thead>
<tr>
<th>peakPermTest</th>
<th>Permutation Test for two given peak lists</th>
</tr>
</thead>
</table>

Description

Performs a permutation test to see if there is an association between two given peak lists.
Usage

peakPermTest(
  peaks1,
  peaks2,
  ntimes = 100,
  seed = as.integer(Sys.time()),
  mc.cores = getOption("mc.cores", 2L),
  maxgap = -1L,
  pool,
  TxDb,
  bindingDistribution,
  bindingType = c("TSS", "geneEnd"),
  featureType = c("transcript", "exon"),
  seqn = NA,
...
)

Arguments

peaks1, peaks2 an object of GRanges
ntimes number of permutations
seed random seed
mc.cores The number of cores to use, see mclapply.
maxgap See findOverlaps in the IRanges package for a description of these arguments.
pool an object of permPool
TxDb an object of TxDb
bindingDistribution an object of bindist
bindingType where the peaks should bind, TSS or geneEnd
featureType what annotation type should be used for detecting the binding distribution.
seqn default is NA, which means not filter the universe pool for sampling. Otherwise
 the universe pool will be filtered by the seqnames in seqn.
... further arguments to be passed to numOverlaps.

Value

A list of class permTestResults. See permTest

Author(s)

Jianhong Ou

References

See Also

    preparePool, bindist

Examples

```r
path <- system.file("extdata", package="ChIPpeakAnno")
#files <- dir(path, pattern="[12]_WS170.bed", full.names=TRUE)
#peaks1 <- toGRanges(files[1], skip=5)
#peaks2 <- toGRanges(files[2], skip=5)
#peakPermTest(peaks1, peaks2, TxDb=TxDB.Celegans.UCSC.ce6.ensGene)
if(interactive()){
    peaks1 <- toGRanges(file.path(path, "MACS2_peaks.xls"),
                        format="MACS2")
    peaks2 <- toGRanges(file.path(path, "peaks.narrowPeak"),
                        format="narrowPeak")
    library(TxDB.Hsapiens.UCSC.hg19.knownGene)
    peakPermTest(peaks1, peaks2,
                 TxDb=TxDB.Hsapiens.UCSC.hg19.knownGene, min.pctA=10)
}
```

Peaks.Ste12.Replicate1

Ste12-binding sites from biological replicate 1 in yeast (see reference)

Description

Ste12-binding sites from biological replicate 1 in yeast (see reference)

Usage

Peaks.Ste12.Replicate1

Format

GRanges with slot names containing the ID of peak as character, slot start containing the start position of the peak, slot end containing the end position of the peak and space containing the chromosome where the peak is located.

References


Examples

```r
data(Peaks.Ste12.Replicate1)
Peaks.Ste12.Replicate1
```
Peaks.Ste12.Replicate2

Ste12-binding sites from biological replicate 2 in yeast (see reference)

Description

Ste12-binding sites from biological replicate 2 in yeast (see reference)

Usage

Peaks.Ste12.Replicate2

Format

GRanges with slot names containing the ID of peak as character, slot start containing the start position of the peak, slot end containing the end position of the peak and space containing the chromosome where the peak is located.

Source

http://www.biomedcentral.com/1471-2164/10/37

References


Examples

data(Peaks.Ste12.Replicate2)
Peaks.Ste12.Replicate2

Peaks.Ste12.Replicate3

Ste12-binding sites from biological replicate 3 in yeast (see reference)

Description

Ste12-binding sites from biological replicate 3 in yeast (see reference)

Usage

Peaks.Ste12.Replicate3
peaks1

Format

GRanges with slot names containing the ID of peak as character, slot start containing the start position of the peak, slot end containing the end position of the peak and space containing the chromosome where the peak is located.

Source

http://www.biomedcentral.com/1471-2164/10/37

References


Examples

data(Peaks.Ste12.Replicate3)
Peaks.Ste12.Replicate3
peaks2
An example GRanges object representing a ChIP-seq peak dataset

Description
An example GRanges object representing a ChIP-seq peak dataset

Usage
peaks2

Format
GRanges

Examples
```r
data(peaks2)
head(peaks2, n = 2)
```

peaks3
An example GRanges object representing a ChIP-seq peak dataset

Description
An example GRanges object representing a ChIP-seq peak dataset

Usage
peaks3

Format
GRanges

Examples
```r
data(peaks3)
head(peaks3, n = 2)
```
peaksNearBDP

obtain the peaks near bi-directional promoters

Description

Obtain the peaks near bi-directional promoters. Also output percent of peaks near bi-directional promoters.

Usage

peaksNearBDP(myPeakList, AnnotationData, MaxDistance = 5000L, ...)

Arguments

myPeakList    GRanges: See example below
AnnotationData annotation data obtained from getAnnotation or customized annotation of class
GRanges containing additional variable: strand (1 or + for plus strand and -1 or - for minus strand). For example, data(TSS.human.NCBi36), data(TSS.mouse.NCBIM37), data(TSS.rat.RGSC3.4) and data(TSS.zebrafish.Zv8).
MaxDistance   Specify the maximum gap allowed between the peak and nearest gene
...           Not used

Value

A list of 4
list("peaksWithBDP")
annotated Peaks containing bi-directional promoters.
GRangesList with slot start holding the start position of the peak, slot end holding the end position of the peak, slot space holding the chromosome location where the peak is located, slot rownames holding the id of the peak. In addition, the following variables are included.
feature: id of the feature such as ensembl gene ID
insideFeature: upstream: peak resides upstream of the feature; downstream: peak resides downstream of the feature; inside: peak resides inside the feature; overlapStart: peak overlaps with the start of the feature; overlapEnd: peak overlaps with the end of the feature; includeFeature: peak include the feature entirely.
distancetoFeature: distance to the nearest feature such as transcription start site. By default, the distance is calculated as the distance between the start of the binding site and the TSS that is the gene start for genes located on the forward strand and the gene end for genes located on the reverse strand. The user can specify the location of peak and location of feature for calculating this feature_range: start and end position of the feature such as gene
feature_strand: 1 or + for positive strand and -1 or - for negative strand where the feature is located
list("percentPeaksWithBDP")
  The percent of input peaks containing bi-directional promoters
list("n.peaks")
  The total number of input peaks
list("n.peaksWithBDP")
  The # of input peaks containing bi-directional promoters

Author(s)
Lihua Julie Zhu, Jianhong Ou

References

See Also
annotatePeakInBatch, findOverlappingPeaks, makeVennDiagram

Examples
if (interactive() || Sys.getenv("USER")=='jianhongou')
{
  data(myPeakList)
  data(TSS.human.NCBI36)
  seqlevelsStyle(TSS.human.NCBI36) <- seqlevelsStyle(myPeakList)
  annotatedBDP = peaksNearBDP(myPeakList[1:6,],
                             AnnotationData=TSS.human.NCBI36,
                             MaxDistance=5000,
                             PeakLocForDistance = "middle",
                             FeatureLocForDistance = "TSS")
  c(annotatedBDP$percentPeaksWithBDP, annotatedBDP$n.peaks,
     annotatedBDP$n.peaksWithBDP)
}

permPool-class

Class "permPool"

Description
An object of class "permPool" represents the possible locations to do permutation test.

Slots
  grs  object of "GRangesList" The list of binding ranges
  N vector of "integer", permutation number for each ranges

permPool-class
**Objects from the Class**

Objects can be created by calls of the form `new("permPool", grs="GRangesList", N="integer")`.

**See Also**

`preparePool`, `peakPermTest`

---

**pie1**

**Pie Charts**

**Description**

Draw a pie chart with percentage

**Usage**

```r
pie1(
  x,
  labels = names(x),
  edges = 200,
  radius = 0.8,
  clockwise = FALSE,
  init.angle = if (clockwise) 90 else 0,
  density = NULL,
  angle = 45,
  col = NULL,
  border = NULL,
  lty = NULL,
  main = NULL,
  percentage = TRUE,
  rawNumber = FALSE,
  digits = 3,
  cutoff = 0.01,
  legend = FALSE,
  legendpos = "topright",
  legendcol = 2,
  radius.innerlabel = radius,
  ...
)
```

**Arguments**

- `x` a vector of non-negative numerical quantities. The values in `x` are displayed as the areas of pie slices.
- `labels` one or more expressions or character strings giving names for the slices. Other objects are coerced by `as.graphicsAnnot`. For empty or NA (after coercion to character) labels, no label nor pointing line is drawn.
edges

the circular outline of the pie is approximated by a polygon with this many edges.

radius

the pie is drawn centered in a square box whose sides range from -1 to 1. If the character strings labeling the slices are long it may be necessary to use a smaller radius.

clockwise

logical indicating if slices are drawn clockwise or counter clockwise (i.e., mathematically positive direction), the latter is default.

init.angle

number specifying the starting angle (in degrees) for the slices. Defaults to 0 (i.e., "3 o’clock") unless clockwise is true where init.angle defaults to 90 (degrees), (i.e., "12 o’clock").

density

the density of shading lines, in lines per inch. The default value of NULL means that no shading lines are drawn. Non-positive values of density also inhibit the drawing of shading lines.

angle

the slope of shading lines, given as an angle in degrees (counter-clockwise).

col

a vector of colors to be used in filling or shading the slices. If missing a set of 6 pastel colours is used, unless density is specified when par("fg") is used.

border, lty

(possibly vectors) arguments passed to polygon which draws each slice.

main

an overall title for the plot.

percentage

logical. Add percentage in the figure or not. default TRUE.

rawNumber

logical. Instead percentage, add raw number in the figure or not. default FALSE.

digits

When set percentage as TRUE, how many significant digits are to be used for percentage. see format. default 3.

cutoff

When percentage is TRUE, if the percentage is lower than cutoff, it will NOT be shown. default 0.01.

legend

logical. Instead of label, draw legend for the pie. default, FALSE.

legendpos, legendcol

legend position and legend columns. see legend

radius.innerlabel

position of percentage or raw number label relative to the circle.

...

graphical parameters can be given as arguments to pie. They will affect the main title and labels only.

Author(s)

Jianhong Ou

See Also

pie

Examples

pie1(1:5)
plotBinOverRegions  plot the coverage of regions

Description

plot the output of binOverRegions or binOverGene

Usage

plotBinOverRegions(dat, ...)

Arguments

dat  A list of matrix which indicate the coverage of regions per bin

...  Parameters could be used by matplot

Author(s)

Jianhong Ou

See Also

binOverRegions, binOverGene

Examples

if(interactive()){
  path <- system.file("extdata", package="ChIPpeakAnno")
  library(TxDb.Hsapiens.UCSC.hg19.knownGene)
  library(rtracklayer)
  files <- dir(path, "bigWig")
  if(.Platform$OS.type != "windows"){
    cvglists <- lapply(file.path(path, files), import,
                       format="BigWig", as="RleList")
    names(cvglists) <- sub(".bigWig", "", files)
    d <- binOverGene(cvglists, TxDb.Hsapiens.UCSC.hg19.knownGene)
    plotBinOverRegions(d)
  }
}
preparePool

prepare data for permutation test

Description

prepare data for permutation test `peakPermTest`

Usage

```r
preparePool(
  TxDb,
  template,
  bindingDistribution,
  bindingType = c("TSS", "geneEnd"),
  featureType = c("transcript", "exon"),
  seqn = NA
)
```

Arguments

- `TxDb`: an object of `TxDb`
- `template`: an object of `GRanges`
- `bindingDistribution`: an object of `bindist`
- `bindingType`: the relevant position to features
- `featureType`: feature type, transcript or exon.
- `seqn`: seqnames. If given, the pool for permutation will be restrict in the given chromosomes.

Value

a list with two elements, `grs`, a list of `GRanges`. `N`, the numbers of elements should be drawn from in each `GRanges`.

Author(s)

Jianhong Ou

See Also

`peakPermTest`, `bindist`
reCenterPeaks

**Examples**

```r
if(interactive() || Sys.getenv("USER")=="jianhongou"){
    path <- system.file("extdata", package="ChIPpeakAnno")
    peaksA <- toGRanges(file.path(path, "peaks.narrowPeak"),
                        format="narrowPeak")
    peaksB <- toGRanges(file.path(path, "MACS2_peaks.xls"), format="MACS2")
    library(TxDb.Hsapiens.UCSC.hg19.knownGene)
    ppp <- preparePool(TxDb.Hsapiens.UCSC.hg19.knownGene,
                        peaksA, bindingType="TSS",
                        featureType="transcript")
}
```

---

**reCenterPeaks**  
**re-center the peaks**

**Description**

Create a new list of peaks based on the peak centers of given list.

**Usage**

```r
reCenterPeaks(peaks, width = 2000L, ...)
```

**Arguments**

- `peaks` An object of GRanges or annoGR.
- `width` The width of new peaks
- `...` Not used.

**Value**

An object of GRanges.

**Author(s)**

Jianhong Ou

**Examples**

```r
reCenterPeaks(GRanges("chr1", IRanges(1, 10)), width=2)
```
summarizeOverlapsByBins

*Perform overlap queries between reads and genomic features by bins*

**Description**

summarizeOverlapsByBins extends `summarizeOverlaps` by providing fixed window size and step to split each feature into bins and then do queries. It will return counts by `signalSummaryFUN`, which applied to bins in one feature, for each feature.

**Usage**

```r
summarizeOverlapsByBins(
  targetRegions,
  reads,
  windowSize = 50,
  step = 10,
  signalSummaryFUN = max,
  mode = countByOverlaps,
  ...
)
```

**Arguments**

- `targetRegions` A GRanges object of genomic regions of interest.
- `reads` A GRanges, GRangesList GAlignments, GAlignmentsList, GAlignmentPairs or BamFileList object that represents the data to be counted by `summarizeOverlaps`.
- `windowSize` Size of windows
- `step` Step of windows
- `signalSummaryFUN` function, which will be applied to the bins in each feature.
- `mode` mode can be one of the pre-defined count methods. see `summarizeOverlaps`. default is `countByOverlaps`, alias of `countOverlaps(features, reads, ignore.strand=ignore.strand)`
- `...` Additional arguments passed to `summarizeOverlaps`.

**Value**

A RangedSummarizedExperiment object. The assays slot holds the counts, rowRanges holds the annotation from features.

**Author(s)**

Jianhong Ou
**Examples**

```r
fls <- list.files(system.file("extdata", package="GenomicAlignments"),
                 recursive=TRUE, pattern="*bam$", full=TRUE)
names(fls) <- basename(fls)
genes <- GRanges(
    seqnames = c(rep("chr2L", 4), rep("chr2R", 5), rep("chr3L", 2)),
    ranges = IRanges(c(1000, 3000, 4000, 7000, 2000, 3000, 3600,
                        4000, 7500, 5000, 5400),
                        width=c(rep(500, 3), 600, 900, 500, 300, 900,
                                300, 500, 500),
                        names=letters[1:11]))
se <- summarizeOverlapsByBins(genes, fls, windowSize=50, step=10)
```

**summarizePatternInPeaks**

*Output a summary of the occurrence and enrichment of each pattern in the sequences.*

**Description**

Output a summary of the occurrence and enrichment of each pattern in the sequences.

**Usage**

```r
summarizePatternInPeaks(
    patternFilePath,
    format = "fasta",
    BSgenomeName,
    peaks,
    revcomp = TRUE,
    method = c("binom.test", "permutation.test"),
    expectFrequencyMethod = c("Markov", "Naive"),
    MarkovOrder = 3L,
    bgdForPerm = c("shuffle", "chromosome"),
    chromosome = c("asPeak", "random"),
    nperm = 1000,
    alpha = 0.05,
    ...
)
```

**Arguments**

- `patternFilePath` Character value. The path to the file that contains the pattern.
- `format` Character value. The format of file containing the oligonucleotide pattern, either "fasta" (default) or "fastq".
summarizePatternInPeaks

BSgenomeName  Character value. BSgenome object. Please refer to available.genomes in BSgenome package for details.

peaks  Character value. GRanges containing the peaks.

revcomp  Boolean value, if TRUE, also search the reverse compliment of pattern. Default is TRUE.

method  Character value. Method for pattern enrichment test, 'binom.test' (default) or 'permutation.test'.

expectFrequencyMethod  Character value. Method for calculating the expected probability of pattern occurrence, 'Markov' (default) or 'Naive'.

MarkovOrder  Integer value. The order of Markov chain. Default is 3.

bgdForPerm  Character value. The method for obtaining the background sequence. 'chromosome' (default) selects background chromosome from chromosomes, refer to 'chromosome' parameter; 'shuffle' will obtain the background sequence by shuffling any k-mers in peak sequences, refer to '...'.

chromosome  Character value. Relevant if 'bgdForPerm='chromosome'". 'asPeak' means to use the same chromosomes in peaks; 'random' means to use all chromosomes randomly. Default is 'asPeak'.

nperm  Integer value. The number of permutation test, default is 1000.

alpha  Numeric value. The significant level for permutation test, default is 0.05.

...  Additional parameter passed to function \link[universalmotif] shuffle_sequences

Details

Please see \link[universalmotif] shuffle_sequences for the more information bout 'shuffle' method.

Value

A list including two data frames named 'motif_enrichment' and 'motif_occurrence'. The 'motif_enrichment' has four columns:

- "patternNum": number of matched pattern
- "totalNumPatternWithSameLen": total number of pattern with the same length
- "expectedRate": expected rate of pattern for 'binom.test' method
- "patternRate": real rate of pattern for 'permutation.test' method
- "pValueBinomTest": p value of binom test for 'binom.test' method
- "cutOffPermutationTest": cut off of permutation test for 'permutation.test' method

The 'motif_occurrence' has 14 columns:

- "motifChr": Chromosome of motif
- "motifStartInChr": motif start position in chromosome
- "motifEndInChr": motif end position in chromosome
- "motifName": motif name
tileCount

Perform overlap queries between reads and genome by windows

Description

tileCount extends summarizeOverlaps by providing fixed window size and step to split whole genome into windows and then do queries. It will return counts in each windows.

Usage

tileCount(
  reads,
  genome,
  windowSize = 1e+06,
  step = 1e+06,
  keepPartialWindow = FALSE,
  mode = countByOverlaps,
...
)
tileGRanges

Arguments

readsa GRanges, GRangesList GAlignments, GAlignmentsList, GAlignmentPairs or BamFileList object that represents the data to be counted by summarizeOverlaps.
genome The object from/on which to get/set the sequence information.
WindowSize Size of windows
step Step of windows
keepPartialWindow Keep last partial window or not.
mode mode can be one of the pre-defined count methods. see summarizeOverlaps. default is countByOverlaps, alias of countOverlaps(features, reads, ignore.strand=ignore.strand)
... Additional arguments passed to summarizeOverlaps.

Value

A RangedSummarizedExperiment object. The assays slot holds the counts, rowRanges holds the annotation from genome.

Author(s)

Jianhong Ou

Examples

```r
fls <- list.files(system.file("extdata", package="GenomicAlignments"),
recursive=TRUE, pattern="*bam$", full=TRUE)
names(fls) <- basename(fls)
genes <- GRanges(seqlengths = c(chr2L=7000, chr2R=10000))
se <- tileCount(fls, genes, windowSize=1000, step=500)
```

tileGRanges  Slide windows on a given GRanges object

Description

tileGRanges returns a set of genomic regions by sliding the windows in a given step. Each window is called a "tile".

Usage

tileGRanges(targetRegions, windowSize, step, keepPartialWindow = FALSE, ...)
toGRanges

Arguments

- targetRegions: A GRanges object of genomic regions of interest.
- windowSize: Size of windows
- step: Step of windows
- keepPartialWindow: Keep last partial window or not.

Value

A GRanges object.

Author(s)

Jianhong Ou

Examples

genes <- GRanges(
    seqnames = c(rep("chr2L", 4), rep("chr2R", 5), rep("chr3L", 2)),
    ranges = IRanges(c(1000, 3000, 4000, 7000, 2000, 3000, 3600,
                       4000, 7500, 5000, 5400),
                      width=c(rep(500, 3), 600, 900, 500, 900, 300, 500, 500),
                      names=letters[1:11])
se <- tileGRanges(genes, windowSize=50, step=10)

Description

Convert UCSC BED format and its variants, such as GFF, or any user defined dataset such as MACS output file to GRanges

Usage

toGRanges(data, ...)

## S4 method for signature 'connection'
toGRanges(
    data,
    format = c("BED", "GFF", "GTF", "MACS", "MACS2", "MACS2.broad", "narrowPeak",
               "broadPeak", "CSV", "others"),
    header = FALSE,
    comment.char = "#",
    ...)
colNames = NULL,
...
)

## S4 method for signature 'TxDb'
toGRanges(
data,
feature = c("gene", "transcript", "exon", "CDS", "fiveUTR", "threeUTR", "microRNA", "tRNAs", "geneModel"),
OrganismDb,
...
)

## S4 method for signature 'EnsDb'
toGRanges(
data,
feature = c("gene", "transcript", "exon", "disjointExons"),
...
)

## S4 method for signature 'character'
toGRanges(
data,
format = c("BED", "GFF", "GTF", "MACS", "MACS2", "MACS2.broad", "narrowPeak", "broadPeak", "CSV", "others"),
header = FALSE,
comment.char = ",",
colNames = NULL,
...
)

Arguments

- **data**: an object of data.frame, TxDb or EnsDb, or the file name of data to be imported. Alternatively, data can be a readable txt-mode connection (See `?read.table`).
- ...
- **format**: data format. If the data format is set to BED, GFF, narrowPeak or broadPeak, please refer to http://genome.ucsc.edu/FAQ/FAQformat#format1 for column order. "MACS" is for converting the excel output file from MACS1. "MACS2" is for converting the output file from MACS2. If set to CSV, must have columns: seqnames, start, end, strand.
- **header**: A logical value indicating whether the file contains the names of the variables as its first line. If missing, the value is determined from the file format: header is set to TRUE if the first row contains one fewer field than the number of columns or the format is set to 'CSV'.
- **comment.char**: character: a character vector of length one containing a single character or an empty string. Use "" to turn off the interpretation of comments altogether.
colNames  
If the data format is set to "others", colname must be defined. And the colname must contain space, start and end. The column name for the chromosome # should be named as space.

feature  
annotation type

OrganismDb  
an object of OrganismDb. It is used for extracting gene symbol for geneModel group for TxDb

Value
An object of GRanges

Author(s)
Jianhong Ou

Examples

```r
macs <- system.file("extdata", "MACS_peaks.xls", package="ChIPpeakAnno")
macsOutput <- toGRanges(macs, format="MACS")
if(interactive() || Sys.getenv("USER")=='jianhongou'){
  ## MACS connection
  macs <- readLines(macs)
  macs <- textConnection(macs)
  macsOutput <- toGRanges(macs, format="MACS")
  close(macs)
  ## bed
  toGRanges(system.file("extdata", "MACS_output.bed", package="ChIPpeakAnno"),
            format="BED")
  ## narrowPeak
  toGRanges(system.file("extdata", "peaks.narrowPeak", package="ChIPpeakAnno"),
            format="narrowPeak")
  ## broadPeak
  toGRanges(system.file("extdata", "TAF.broadPeak", package="ChIPpeakAnno"),
            format="broadPeak")
  ## CSV
  toGRanges(system.file("extdata", "peaks.csv", package="ChIPpeakAnno"),
            format="CSV")
  ## MACS2
  toGRanges(system.file("extdata", "MACS2_peaks.xls", package="ChIPpeakAnno"),
            format="MACS2")
  ## GFF
  toGRanges(system.file("extdata", "GFF_peaks.gff", package="ChIPpeakAnno"),
            format="GFF")
  ## EnsDb
  library(EnsDb.Hsapiens.v75)
  toGRanges(EnsDb.Hsapiens.v75, feature="gene")
  ## TxDb
  library(TxDb.Hsapiens.UCSC.hg19.knownGene)
  toGRanges(TxDb.Hsapiens.UCSC.hg19.knownGene, feature="gene")
  ## data.frame
  macs <- system.file("extdata", "MACS_peaks.xls", package="ChIPpeakAnno")
```
translatePattern

translate pattern from IUPAC Extended Genetic Alphabet to regular expression

Description

translate pattern containing the IUPAC nucleotide ambiguity codes to regular expression. For example, Y->[C|T], R-> [A|G], S-> [G|C], W-> [A|T], K-> [T|U|G], M-> [A|C], B-> [G|T], D-> [A|G|T], H-> [A|C|T], V-> [A|C|G] and N-> [A|C|T|G].

Usage

translatePattern(pattern)

Arguments

pattern a character vector with the IUPAC nucleotide ambiguity codes

Value

a character vector with the pattern represented as regular expression

Author(s)

Lihua Julie Zhu

See Also

countPatternInSeqs, summarizePatternInPeaks

Examples

pattern1 = "AACCNWKM"
translatePattern(pattern1)
TSS.human.GRCh37

TSS annotation for human sapiens (GRCh37) obtained from biomaRt

Description
TSS annotation for human sapiens (GRCh37) obtained from biomaRt

Usage
TSS.human.GRCh37

Format
A GRanges object with slot start holding the start position of the gene, slot end holding the end position of the gene, slot names holding ensembl gene id, slot seqnames holding the chromosome location where the gene is located and slot strand holding the strand information. In addition, the following variables are included.

list("description") description of the gene

Details
The dataset TSS.human.GRCh37 was obtained by:
mart = useMart(biomart = "ENSEMBL_MART_ENSEMBL", host="grch37.ensembl.org", path="/biomart/martservice", dataset = "hsapiens_gene_ensembl")
getAnnotation(mart, featureType = "TSS")

Examples
data(TSS.human.GRCh37)
slotNames(TSS.human.GRCh37)

TSS.human.GRCh38

TSS annotation for human sapiens (GRCh38) obtained from biomaRt

Description
TSS annotation for human sapiens (GRCh38) obtained from biomaRt

Usage
TSS.human.GRCh38

Format
A 'GRanges' [package "GenomicRanges"] object with ensembl id as names.
TSS.human.NCBI36

Details

used in the examples Annotation data obtained by:

```r
mart = useMart(biomart = "ensembl", dataset = "hsapiens_gene_ensembl")
getAnnotation(mart, featureType = "TSS")
```

Examples

```r
data(TSS.human.GRCh38)
slotNames(TSS.human.GRCh38)
```

---

TSS.human.NCBI36  TSS annotation for human sapiens (NCBI36) obtained from biomaRt

Description

TSS annotation for human sapiens (NCBI36) obtained from biomaRt

Usage

TSS.human.NCBI36

Format

GRanges with slot start holding the start position of the gene, slot end holding the end position of the gene, slot names holding ensembl gene id, slot seqnames holding the chromosome location where the gene is located and slot strand holding the strand information. In addition, the following variables are included.

- `list("description")`: description of the gene

Details

used in the examples Annotation data obtained by:

```r
mart = useMart(biomart = "ensembl_mart_47", dataset = "hsapiens_gene_ensembl", archive=TRUE)
getAnnotation(mart, featureType = "TSS")
```

Examples

```r
data(TSS.human.NCBI36)
slotNames(TSS.human.NCBI36)
```
TSS.mouse.GRCm38  

TSS annotation data for Mus musculus (GRCm38.p1) obtained from biomaRt

Description

TSS annotation data for Mus musculus (GRCm38.p1) obtained from biomaRt

Usage

TSS.mouse.GRCm38

Format

GRanges with slot start holding the start position of the gene, slot end holding the end position of the gene, slot names holding ensembl gene id, slot seqnames holding the chromosome location where the gene is located and slot strand holding the strand information. In addition, the following variables are included.

list("description")  description of the gene

Details

Annotation data obtained by:

```r
mart = useMart(biomart = "ensembl", dataset = "mmusculus_gene_ensembl")
getAnnotation(mart, featureType = "TSS")
```

Examples

```r
data(TSS.mouse.GRCm38)
slotNames(TSS.mouse.GRCm38)
```

TSS.mouse.NCBIM37  

TSS annotation data for mouse (NCBIM37) obtained from biomaRt

Description

TSS annotation data for mouse (NCBIM37) obtained from biomaRt

Usage

TSS.mouse.NCBIM37
Format

GRanges with slot start holding the start position of the gene, slot end holding the end position of the gene, slot names holding ensembl gene id, slot seqnames holding the chromosome location where the gene is located and slot strand holding the strinad information. In addition, the following variables are included.

list("description")  description of the gene

Details

Annotation data obtained by:

```r
mart = useMart(biomart = "ensembl", dataset = "mmusculus_gene_ensembl")
getAnnotation(mart, featureType = "TSS")
```

Examples

```r
data(TSS.mouse.NCBIM37)
slotNames(TSS.mouse.NCBIM37)
```

<table>
<thead>
<tr>
<th>TSS.rat.RGSC3.4</th>
<th>TSS annotation data for rat (RGSC3.4) obtained from biomaRt</th>
</tr>
</thead>
</table>

Description

TSS annotation data for rat (RGSC3.4) obtained from biomaRt

Usage

TSS.rat.RGSC3.4

Format

GRanges with slot start holding the start position of the gene, slot end holding the end position of the gene, slot names holding ensembl gene id, slot seqnames holding the chromosome location where the gene is located and slot strand holding the strinad information. In addition, the following variables are included.

list("description")  description of the gene

Details

Annotation data obtained by:

```r
mart = useMart(biomart = "ensembl", dataset = "rnorvegicus_gene_ensembl")
getAnnotation(mart, featureType = "TSS")
```

Examples

```r
data(TSS.rat.RGSC3.4)
slotNames(TSS.rat.RGSC3.4)
```
**TSS.rat.Rnor_5.0**

**Description**

TSS annotation data for Rattus norvegicus (Rnor_5.0) obtained from biomaRt

**Usage**

TSS.rat.Rnor_5.0

**Format**

GRanges with slot start holding the start position of the gene, slot end holding the end position of the gene, slot names holding ensembl gene id, slot seqnames holding the chromosome location where the gene is located and slot strand holding the strinad information. In addition, the following variables are included.

- `list("description")`: description of the gene

**Details**

Annotation data obtained by:

- `mart = useMart(biomart = "ensembl", dataset = "rnorvegicus_gene_ensembl")`
- `getAnnotation(mart, featureType = "TSS")`

**Examples**

- `data(TSS.rat.Rnor_5.0)`
- `slotNames(TSS.rat.Rnor_5.0)`

---

**TSS.zebrafish.Zv8**

**Description**

A GRanges object to annotate TSS for zebrafish (Zv8) obtained from biomaRt

**Usage**

TSS.zebrafish.Zv8
Format

GRanges with slot start holding the start position of the gene, slot end holding the end position of the gene, slot names holding ensembl gene id, slot seqnames holding the chromosome location where the gene is located and slot strand holding the strinad information. In addition, the following variables are included.

\textbf{list("description")} description of the gene

Details

Annotation data obtained by:

\begin{verbatim}
mart <- useMart(biomart="ENSEMBL\_MART\_ENSEMBL", host="may2009.archive.ensembl.org", path="/biomart/martservice", dataset="drerio\_gene\_ensembl")
getAnnotation(mart, featureType = "TSS")
\end{verbatim}

Examples

\begin{verbatim}
data(TSS.zebrafish.Zv8)
slotNames(TSS.zebrafish.Zv8)
\end{verbatim}

TSS.zebrafish.Zv9  \textit{TSS annotation for Danio rerio (Zv9) obtained from biomaRt}
**Examples**

```r
data(TSS.zebrafish.Zv9)
slotNames(TSS.zebrafish.Zv9)
```

---

**Description**

:convert TxDb object to GRanges

**Usage**

```r
TxDb2GR(ranges, feature, OrganismDb)
```

**Arguments**

- **ranges**: an Txdb object
- **feature**: feature type, could be `geneModel`, `gene`, `exon`, `transcript`, `CDS`, `fiveUTR`, `threeUTR`, `microRNA`, and `tRNA`
- **OrganismDb**: org db object

---

**wgEncodeTfbsV3**

*transcription factor binding site clusters (V3) from ENCODE*

---

**Description**

: possible binding pool for human (hg19) from transcription factor binding site clusters (V3) from ENCODE data and removed the HOT spots

**Usage**

```r
wgEncodeTfbsV3
```

**Format**

An object of GRanges.
Details

How to generate the data:

```r
temp <- tempfile()
download.file(file.path("http://hgdownload.cse.ucsc.edu", "goldenPath", "hg19", "encodeDCC", "wgEncodeRegTfbsClustered", "wgEncodeRegTfbsClusteredV3.bed.gz"), temp)
data <- read.delim(gzfile(temp, "r"), header=FALSE)
unlink(temp)
colnames(data)[1:4] <- c("seqnames", "start", "end", "TF")
wgEncodeRegTfbsClusteredV3 <- GRanges(as.character(data$seqnames), IRanges(data$start, data$end), TF=data$TF)
data(HOT.spots)
hot <- reduce(unlist(HOT.spots))
ol <- findOverlaps(wgEncodeRegTfbsClusteredV3, hot)
wgEncodeTfbsV3 <- wgEncodeRegTfbsClusteredV3[-unique(queryHits(ol))]
wgEncodeTfbsV3 <- reduce(wgEncodeTfbsV3)
save(list="wgEncodeTfbsV3", file="data/wgEncodeTfbsV3.rda", compress="xz", compression_level=9)
```

Source

http://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/ wgEncodeRegTfbsClustered/wgEncodeRegTfbsClusteredV3.bed.gz

Examples

```r
data(wgEncodeTfbsV3)
head(wgEncodeTfbsV3)
```
write2FASTA

Write sequences to a file in fasta format

Description

Write the sequences obtained from getAllPeakSequence to a file in fasta format leveraging write-FASTA in Biostrings package. FASTA is a simple file format for biological sequence data. A FASTA format file contains one or more sequences and there is a header line which begins with a > proceeding each sequence.

Usage

write2FASTA(mySeq, file = "", width = 80)

Arguments

mySeq  GRanges with variables name and sequence , e.g., results obtained from getAllPeakSequence
file Either a character string naming a file or a connection open for reading or writing. If "" (the default for write2FASTA), then the function writes to the standard output connection (the console) unless redirected by sink
width The maximum number of letters per line of sequence

Value

Output as FASTA file format to the naming file or the console.

Author(s)

Lihua Julie Zhu

Examples

peaksWithSequences = GRanges(seqnames=c("1", "2"),
IRanges(start=c(1000, 2000),
end=c(1010, 2010),
names=c("id1", "id2")),
sequence= c("CCCCCCCGGGGG", "TTTTTTTTAAAAA"))

write2FASTA(peaksWithSequences, file="testseq.fasta", width=50)
Description
Search by name for an Bimap object.

Usage
xget(
  x,
  envir,
  mode,
  ifnotfound = NA,
  inherits,
  output = c("all", "first", "last")
)

Arguments
x, envir, mode, ifnotfound, inherits
  see mget
output
  return the all or first item for each query

Value
a character vector

Author(s)
Jianhong Ou

See Also
See Also as mget, mget

Examples
library(org.Hs.eg.db)
xget(as.character(1:10), org.Hs.egSYMBOL)
Index

* classes
  annoGR-class, 10
  bindist-class, 24
  permPool-class, 90

* datasets
  annotatedPeak, 13
  enrichedGO, 35
  ExonPlusUtr.human.GRCh37, 41
  HOT.spots, 72
  myPeakList, 81
  Peaks.Ste12.Replicate1, 85
  Peaks.Ste12.Replicate2, 86
  Peaks.Ste12.Replicate3, 86
  peaks1, 87
  peaks2, 88
  peaks3, 88
  TSS.human.GRCh37, 105
  TSS.human.GRCh38, 105
  TSS.human.NCBI36, 106
  TSS.mouse.GRCm38, 107
  TSS.mouse.NCBI37, 107
  TSS.rat.RGSC3.4, 108
  TSS.rat.Rnor_5.0, 109
  TSS.zebrafish.Zv8, 109
  TSS.zebrafish.Zv9, 110
  wgEncodeTfbsV3, 111

* graph
  makeVennDiagram, 76

* internal
  getGeneSeq, 67
  getUniqueGOidCount, 69
  hyperGtest, 73

* misc
  addAncestors, 6
  addGeneIDs, 7
  addMetadata, 9
  annoPeaks, 11
  annotatePeakInBatch, 14
  assignChromosomeRegion, 21
  bdp, 23
  binOverFeature, 25
  condenseMatrixByColnames, 30
  convert2EntrezID, 31
  countPatternInSeqs, 32
  egOrgMap, 35
  estFragmentLength, 38
  estLibSize, 40
  featureAlignedDistribution, 42
  featureAlignedExtendSignal, 43
  featureAlignedHeatmap, 45
  featureAlignedSignal, 46
  findEnhancers, 47
  findOverlappingPeaks, 52
  findOverlapsOfPeaks, 54
  getAllPeakSequence, 59
  getAnnotation, 61
  getEnrichedGO, 62
  getEnrichedPATH, 65
  getGeneSeq, 67
  getGO, 68
  getVennCounts, 70
  IDRfilter, 74
  mergePlusMinusPeaks, 78
  oligoFrequency, 81
  oligoSummary, 82
  peakPermTest, 83
  peaksNearBDP, 89
  pie1, 91
  preparePool, 94
  reCenterPeaks, 95
  summarizeOverlapsByBins, 96
  summarizePatternInPeaks, 97
  tileCount, 99
  tileGRanges, 100
  toGRanges, 101
  translatePattern, 104
  write2FASTA, 113
  xget, 114
package

ChIPpeakAnno-package, 4
bindist-method(bindist-class), 24
permPool-method(permPool-class), 90
bindist-method(bindist-class), 24
permPool-method(permPool-class), 90
acf, 39
addAncestors, 6
addGeneIDs, 7, 17
annoGR, 15, 17, 24, 25, 95
annoGR(annoGR-class), 10
annoGR,EnDb-method(annoGR-class), 10
annoGR,GRanges-method(annoGR-class), 10
annoGR,TxDb-method(annoGR-class), 10
annoGR-class, 10
annoPeaks, 11, 15–17, 24
annotatedPeak, 13
annotatePeakInBatch, 13, 14, 49, 56
assignChromosomeRegion, 21
BamFileList, 96, 100
bdp, 23
bindist, 84, 85, 94
bindist-method(bindist-class), 24
bindist-class, 24
binOverFeature, 23, 25
binOverGene, 23, 26, 28, 93
binOverRegions, 23, 27, 29, 93
ChIPpeakAnno(ChIPpeakAnno-package), 4
ChIPpeakAnno-deprecated, 29
ChIPpeakAnno-package, 4
cntOverlaps, 30
core(annoGR-class), 10
core,annoGR,GRanges-method(annoGR-class), 10
core,GRanges,annoGR-method(annoGR-class), 10
condenseMatrixByColnames, 30
convert2EntrezID, 31
countPatternInSeqs, 32
cumulativePercentage, 33
Date, 10, 11
Deprecation, 30
downstreams, 34
egOrgMap, 35
enrichedGO, 35
enrichmentPlot, 37
EnsDb, 10, 11, 102
EnsDb2GR, 38
estFragmentLength, 38, 44
estLibSize, 40, 44
ExonPlusUtr.human.GRCh37, 41
featureAlignedDistribution, 42, 46, 47
featureAlignedExtendSignal, 43
featureAlignedHeatmap, 42, 45, 47
featureAlignedSignal, 42, 44–46, 46
findEnhancers, 47
findMotifsInPromoterSeqs, 49
findOverlappingPeaks, 17, 52, 56, 71
findOverlappingPeaks-deprecated(findOverlappingPeaks), 52
findOverlaps, 29, 52, 54, 71, 75, 76, 84
findOverlapsOfPeaks, 9, 29, 30, 53, 54, 77
format, 92
frequency, 82, 83
GAInteractions, 96, 100
GAlignments, 96, 100
GAlignmentsList, 96, 100
genomicElementDistribution, 23, 56
genomicElementUpSetR, 23, 56
getAllPeakSequence, 59, 81, 82
getAnnotation, 17, 61
getBM, 8
getEnrichedGO, 37, 62
getEnrichedPATH, 37, 65
geneSeq, 67
gEO, 68
getUniqueGOidCount, 69
getVennCounts, 56, 70
GRanges, 10–12, 15, 17, 24, 25, 29, 33, 42, 43, 45, 47, 48, 53–55, 57, 59–61, 71, 75, 76, 80, 84, 89, 94–96, 98, 100, 101, 103
GRangesList, 57, 59, 80, 96, 100
HOT.spots, 72
hyperGtest, 73
IDRfilter, 74
import, 48
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>importGInteractions</td>
<td>48</td>
</tr>
<tr>
<td>info (annoGR-class)</td>
<td>10</td>
</tr>
<tr>
<td>info,annoGR-method (annoGR-class)</td>
<td>10</td>
</tr>
<tr>
<td>legend</td>
<td>92</td>
</tr>
<tr>
<td>listAttributes(mart)</td>
<td>8</td>
</tr>
<tr>
<td>listFilters(mart)</td>
<td>8</td>
</tr>
<tr>
<td>makeVennDiagram</td>
<td>17, 56, 71, 76</td>
</tr>
<tr>
<td>matplot</td>
<td>42, 93</td>
</tr>
<tr>
<td>mergePlusMinusPeaks</td>
<td>78</td>
</tr>
<tr>
<td>metagenePlot</td>
<td>80</td>
</tr>
<tr>
<td>mget</td>
<td>114</td>
</tr>
<tr>
<td>myPeakList</td>
<td>81</td>
</tr>
<tr>
<td>numOverlaps</td>
<td>30, 84</td>
</tr>
<tr>
<td>oligoFrequency</td>
<td>81</td>
</tr>
<tr>
<td>oligoSummary</td>
<td>82, 82</td>
</tr>
<tr>
<td>OrganismDb</td>
<td>103</td>
</tr>
<tr>
<td>overlappingPeaks</td>
<td>9</td>
</tr>
<tr>
<td>overlappingPeaks (findOverlapsOfPeaks)</td>
<td>54</td>
</tr>
<tr>
<td>overlappingPeaks-class</td>
<td>(findOverlapsOfPeaks), 54</td>
</tr>
<tr>
<td>peakPermTest</td>
<td>25, 77, 83, 91, 94</td>
</tr>
<tr>
<td>Peaks.Ste12.Replicate1</td>
<td>85</td>
</tr>
<tr>
<td>Peaks.Ste12.Replicate2</td>
<td>86</td>
</tr>
<tr>
<td>Peaks.Ste12.Replicate3</td>
<td>86</td>
</tr>
<tr>
<td>peaks1</td>
<td>87</td>
</tr>
<tr>
<td>peaks2</td>
<td>88</td>
</tr>
<tr>
<td>peaks3</td>
<td>88</td>
</tr>
<tr>
<td>peaksNearBDP</td>
<td>17, 89</td>
</tr>
<tr>
<td>permPool</td>
<td>84</td>
</tr>
<tr>
<td>permPool (permPool-class)</td>
<td>90</td>
</tr>
<tr>
<td>permPool-class</td>
<td>90</td>
</tr>
<tr>
<td>permPool-method (permPool-class)</td>
<td>90</td>
</tr>
<tr>
<td>permTest</td>
<td>84</td>
</tr>
<tr>
<td>pie</td>
<td>92</td>
</tr>
<tr>
<td>piel</td>
<td>91</td>
</tr>
<tr>
<td>plotBinOverRegions</td>
<td>27, 28, 93</td>
</tr>
<tr>
<td>preparePool</td>
<td>25, 85, 91, 94</td>
</tr>
<tr>
<td>RangedSummarizedExperiment</td>
<td>96, 100</td>
</tr>
<tr>
<td>read.table</td>
<td>102</td>
</tr>
<tr>
<td>reCenterPeaks</td>
<td>95</td>
</tr>
<tr>
<td>RleList</td>
<td>27, 28, 42, 45, 47</td>
</tr>
<tr>
<td>shuffle_sequences</td>
<td>98</td>
</tr>
<tr>
<td>SimpleRleList</td>
<td>27, 28, 42, 45, 47</td>
</tr>
<tr>
<td>summarizeOverlaps</td>
<td>33, 96, 99, 100</td>
</tr>
<tr>
<td>summarizeOverlapsByBins</td>
<td>96</td>
</tr>
<tr>
<td>summarizePatternInPeaks</td>
<td>17, 97</td>
</tr>
<tr>
<td>tileCount</td>
<td>99</td>
</tr>
<tr>
<td>tileGRanges</td>
<td>100</td>
</tr>
<tr>
<td>toGRanges</td>
<td>30, 101</td>
</tr>
<tr>
<td>toGRanges, character-method (toGRanges)</td>
<td>101</td>
</tr>
<tr>
<td>toGRanges, connection-method (toGRanges)</td>
<td>101</td>
</tr>
<tr>
<td>toGRanges, data.frame-method (toGRanges)</td>
<td>101</td>
</tr>
<tr>
<td>toGRanges, EnDb-method (toGRanges)</td>
<td>101</td>
</tr>
<tr>
<td>toGRanges, TxDb-method (toGRanges)</td>
<td>101</td>
</tr>
<tr>
<td>translatePattern</td>
<td>104</td>
</tr>
<tr>
<td>TSS.human.GRCh37</td>
<td>105</td>
</tr>
<tr>
<td>TSS.human.GRCh38</td>
<td>105</td>
</tr>
<tr>
<td>TSS.human.NCBI36</td>
<td>106</td>
</tr>
<tr>
<td>TSS.mouse.GRCm38</td>
<td>107</td>
</tr>
<tr>
<td>TSS.mouse.NCBI37</td>
<td>107</td>
</tr>
<tr>
<td>TSS.rat.RGSC3.4</td>
<td>108</td>
</tr>
<tr>
<td>TSS.rat.Rnor_5.0</td>
<td>109</td>
</tr>
<tr>
<td>TSS.zebrafish.Zv8</td>
<td>109</td>
</tr>
<tr>
<td>TSS.zebrafish.Zv9</td>
<td>110</td>
</tr>
<tr>
<td>TxDb, 10, 11, 21, 22, 27, 28, 57, 59, 77, 80, 84, 94, 102, 103</td>
<td></td>
</tr>
<tr>
<td>TxDb2GR</td>
<td>111</td>
</tr>
<tr>
<td>useMart</td>
<td>7</td>
</tr>
<tr>
<td>venn.diagram</td>
<td>77</td>
</tr>
<tr>
<td>wgEncodeTfbsV3</td>
<td>111</td>
</tr>
<tr>
<td>write2FASTA</td>
<td>113</td>
</tr>
<tr>
<td>xget</td>
<td>114</td>
</tr>
</tbody>
</table>