Package 'ChIPpeakAnno'

October 17, 2017

Type Package

Title Batch annotation of the peaks identified from either ChIP-seq, ChIP-chip experiments or any experiments resulted in large number of chromosome ranges

Version 3.10.2

Date 2017-10-04

Author Lihua Julie Zhu, Jianhong Ou, Jun Yu, Herve Pages, Claude Gazin, Nathan Lawson, Ryan Thompson, Simon Lin, David Lapointe and Michael Green

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

Depends R (>= 3.2), methods, grid, IRanges (>= 2.5.27), Biostrings, GenomicRanges (>= 1.23.16), S4Vectors (>= 0.9.25), VennDiagram

Imports BiocGenerics (>= 0.1.0), GO.db, biomaRt, BSgenome, GenomicFeatures, GenomeInfoDb, matrixStats, AnnotationDbi, limma, multtest, RBGL, graph, BiocInstaller, stats, regioneR, DBI, ensembldb, Biobase, seqinr, idr, GenomicAlignments, DelayedArray, SummarizedExperiment, Rsamtools

Suggests reactome.db, BSgenome.Ecoli.NCBI.20080805, BSgenome.Hsapiens.UCSC.hg19, org.Ce.eg.db, org.Hs.eg.db, BSgenome.Celegans.UCSC.ce10, BSgenome.Drerio.UCSC.danRer7, EnsDb.Hsapiens.v75, EnsDb.Hsapiens.v79, TxDb.Hsapiens.UCSC.hg19.knownGene, TxDb.Hsapiens.UCSC.hg38.knownGene, gplots, BiocStyle, rtracklayer, knitr, rmarkdown, BiocStyle, testthat, trackViewer, motifStack, OrganismDbi

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 supplied by users. Starting 2.0.5, new functions have been added for finding the peaks with bi-directional promoters with summary statistics (peaksNearBDP), for summarizing the occurrence of motifs in peaks (summarizePatternInPeaks) and for adding other IDs to annotated peaks or enrichedGO (addGeneIDs). This package leverages the biomaRt, IRanges, Biostrings, BSgenome, GO.db, multtest and stat packages.

License GPL (>= 2)

LazyLoad yes

biocViews Annotation, ChIPSeq, ChIPchip

VignetteBuilder knitr

RoxygenNote 5.0.1

NeedsCompilation no

R topics documented:

`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, Herve Pages, 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

1. Y. Benjamini and Y. Hochberg (1995). Controlling the false discovery rate: a practical and powerful approach to multiple testing. J. R. Statist. Soc. B. Vol. 57: 289-300.
2. Y. Benjamini and D. Yekutieli (2001). The control of the false discovery rate in multiple hypothesis testing under dependency. Annals of Statistics. Accepted.
3. S. Durinck et al. (2005) BioMart and Bioconductor: a powerful link between biological biomarts and microarray data analysis. Bioinformatics, 21, 3439-3440.
4. S. Dudoit, J. P. Shaffer, and J. C. Boldrick (Submitted). Multiple hypothesis testing in microarray experiments.
5. Y. Ge, S. Dudoit, and T. P. Speed. Resampling-based multiple testing for microarray data hypothesis, Technical Report #633 of UCB Stat. http://www.stat.berkeley.edu/~gyc
6. Y. Hochberg (1988). A sharper Bonferroni procedure for multiple tests of significance, Biometrika. Vol. 75: 800-802.
7. S. Holm (1979). A simple sequentially rejective multiple test procedure. Scand. J. Statist.. Vol. 6: 65-70.
8. N. L. Johnson,S. Kotz and A. W. Kemp (1992) Univariate Discrete Distributions, Second Edition. New York: Wiley
9. Zhu L.J. et al. (2010) ChIPpeakAnno: a Bioconductor package to annotate ChIP-seq and ChIP-chip data. BMC Bioinformatics 2010, 11:237doi:10.1186/1471-2105-11-237.

Examples

```
if(interactive()){
 data(myPeakList)
 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 package

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"))
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

```
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
  - 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, AnnotationDbi

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,
            IDs2Add=c("hgnc_symbol","entrezgene"))
}
```

`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(1549800,1554400,1565000,1569400,167888600),
                           end=c(1550599,1560799,1565399,1571199,167888999),
                           names=c("f1","f2","f3","f4","f5")),
                   strand="+",
                   score=6:10, id=LETTERS[1:5])
ol <- findOverlapsOfPeaks(peaks1, peaks2)
addMetadata(ol)
```

---

annoGR-class *Class* annoGR

Description

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

Usage

```
    ## 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

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

Objects from the Class

Objects can be created by calls of the form `new("annoGR", date, elementMetadata, feature, mdata, ranges,`

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

Coercion

as(from, "annoGR"): Creates a annoGR object from a GRanges object.

as(from, "GRanges"): Create a GRanges object from a annoGR object.

Methods

info Print basic info for annoGR object

annoGR("TxDb"), annoGR("EnsDb") Create a annoGR object from TxDb or EnsDb object

Author(s)

Jianhong Ou

Examples

```
if(interactive()){
    library(EnsDb.Hsapiens.v79)
    anno <- annoGR(EnsDb.Hsapiens.v79)
}
```

---

annoPeaks *Annotate peaks*

Description

Annotate peaks by annoGR object in the given range.

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 sec ond 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.
...	Not used.

Value

Output is a GRanges object of the annotated peaks.

Author(s)

Jianhong Ou

See Also

See Also as annotatePeakInBatch

Examples

```
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

```
data(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.

`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

`start_position` start position of the feature such as gene

`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

```
data(annotatedPeak)
str(annotatedPeak)
if (interactive()) {
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

```
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=0L, PeakLocForDistance=c("start", "middle", "end"),
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 charcter vector used with mart argument if AnnotationData is not supplied; it's value is "TSS"", "miRNA"" or "Exon"
AnnotationData	A GRanges or annoGR oject. It can be obtained from function getAnnotation or customized annotation of class GRanges containing additional variable: strand (1 or + for plus strand and -1 or - for minus strand). Pre-compliled 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 real time by using the parameters of mart and featureType
output	**nearestLocation (default)** will output the nearest features calculated as PeakLoc - FeatureLocForDistance **overlapping** will output overlapping features with maximum gap specified as maxgap between peak range and feature range **shortestDistance** will output nearest features **both** will output all the nearest features, in addition, will output any features that overlap the peak that is not the nearest features **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.

`maxgap` Non-negative integer. Intervals with a separation of maxgap or less are considered to be overlapping

`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, 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.

`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 sec ond 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)

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.

`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
`start_position`	start position of the feature such as gene
`end_position`	end position of the feature such as the gene
`strand`	1 or + for positive strand and -1 or - for negative strand where the feature is located
`shortestDistance`	The shortest distance from either end of peak to either end the feature.
`fromOverlappingOrNearest`	nearest: indicates this feature's start (feature's end for features at minus strand) is closest to the peak start; Overlapping: indicates this feature overlaps with this peak although it is not the nearest feature start

Author(s)

Lihua Julie Zhu, Jianhong Ou

References

1. Zhu L.J. et al. (2010) ChIPpeakAnno: a Bioconductor package to annotate ChIP-seq and ChIP-chip data. BMC Bioinformatics 2010, 11:237doi:10.1186/1471-2105-11-237

2. Zhu L (2013). "Integrative analysis of ChIP-chip and ChIP-seq dataset." In Lee T and Luk ACS (eds.), Tilling Arrays, volume 1067, chapter 4, pp. -19. Humana Press. http://dx.doi.org/10.1007/978-1-62703-607-8_8

See Also

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

Examples

```
#if (interactive()){
    ## example 1: annotate myPeakList by TxDb or EnsDb.
    data(myPeakList)
    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 biding 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 RangedData
## 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("+", "-"), 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
test.bed <- data.frame(space=c("4", "6"),
                       start=c("100", "1000"),
                       end=c("200", "1100"),
                       name=c("peak1", "peak2"))
test.GR = toGRanges(test.bed)
annotatePeakInBatch(test.GR, AnnotationData = literature)
#}
```

`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=1000L, immediate.downstream.cutoff=1000L,
        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 enhencer. 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 1000 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 prime UTR are classified as immediate downstream. Peaks that reside downstream over immediate.downstreatm.cutoff from gene end are classified as enhancers. The default is 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 conbinations 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:

Exons	Percent or the picard index of the peaks resided in exon regions.
Introns	Percent or the picard index of the peaks resided in intron regions.
fiveUTRs	Percent or the picard index of the peaks resided in 5 prime UTR regions.
threeUTRs	Percent or the picard index of the peaks resided in 3 prime UTR regions.
Promoter	Percent or the picard index of the peaks resided in proximal promoter regions.
ImmediateDownstream	Percent or the picard index of the peaks resided in immediate downstream regions.
Enhancer.Silencer	Percent or the picard index of the peaks resided in enhancer/silencer regions.

Author(s)

Jianhong Ou, Lihua Julie Zhu

References

1. Zhu L.J. et al. (2010) ChIPpeakAnno: a Bioconductor package to annotate ChIP-seq and ChIP-chip data. BMC Bioinformatics 2010, 11:237doi:10.1186/1471-2105-11-237

2. Zhu L.J. (2013) Integrative analysis of ChIP-chip and ChIP-seq dataset. Methods Mol Biol. 2013;1067:105-24. doi: 10.1007/978-1-62703-607-8_8.

See Also

annotatePeakInBatch, findOverlapsOfPeaks,getEnriched, makeVennDiagram,addGeneIDs, peaksNearBDP,summarizePa

Examples

```
if (interactive()){
    ##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

```
if(interactive()){
  library(EnsDb.Hsapiens.v75)
  data("myPeakList")
  annoGR <- annoGR(EnsDb.Hsapiens.v75)
  seqlevelsStyle(myPeakList) <- seqlevelsStyle(annoGR)
  bdp(myPeakList, annoGR)
}
```

BED2RangedData *Convert BED format to RangedData*

Description

Convert BED format to RangedData. This function will be depreciated.

Usage

```
BED2RangedData(data.BED,header=FALSE, ...)
```

Arguments

data.BED	BED format data frame or BED filename, please refer to http://genome.ucsc.edu/FAQ/FAQformat#for for details
header	TRUE or FALSE, default to FALSE, indicates whether data.BED file has BED header
...	any parameter need to be passed into read.delim function

Value

RangedData 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.

strand	1 for positive strand and -1 for negative strand where the feature is located. Default to 1 if not present in the BED formated data frame

Note

For converting the peakList in BED format to RangedData before calling annotatePeakInBatch function

Author(s)

Lihua Julie Zhu

See Also

See also as toGRanges.

Examples

```
test.bed = data.frame(cbind(chrom = c("1", "2"),
                      chromStart=c("100", "1000"),
                      chromEnd=c("200", "1100"),
                      name=c("peak1", "peak2")))
test.rangedData = BED2RangedData(test.bed)
```

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="in

Slots

counts vector of "integer" The count number in each binding range

mids vector of "integer" The center of each range relevant to feature

halfBinSize "integer", length must be 1. the fixed half-width of each binding range

bindingType a "character". could be "TSS", "geneEnd"

featureType a "character". could be "transcript", "exon"

Methods

$, $<- Get or set the slot of bindist

See Also

preparePool, peakPermTest

binOverFeature *Aggregate peaks over bins from the TSS*

Description

Aggregate peaks over bins from the feature sites.

Usage

```
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
PeakLocForDistance	which site of peaks should be used for distance calculation
FUN	the function to be used for score calculation
errFun	the function to be used for errorbar calculation or values for the errorbar.
xlab	titles for each x axis
ylab	titles for each y axis
main	overall titles for each plot

Value

A data.frame with bin values.

Author(s)

Jianhong Ou

Examples

```
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)
```

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.

Usage

```
findOverlappingPeaks(Peaks1, Peaks2, maxgap = 0L,
                  minoverlap=1L, multiple = c(TRUE, FALSE),
                  NameOfPeaks1 = "TF1", NameOfPeaks2 = "TF2",
                  select=c("all", "first","last","arbitrary"),
                  annotate = 0, ignore.strand=TRUE,
                  connectedPeaks=c("min", "merge"), ...)
BED2RangedData(data.BED,header=FALSE, ...)
GFF2RangedData(data.GFF,header=FALSE, ...)
```

Arguments

Peaks1	RangedData: See example below.
Peaks2	RangedData: See example below.
maxgap	Non-negative integer. Intervals with a separation of maxgap or less are considered to be overlapping.
minoverlap	Non-negative integer. Intervals with an overlapping of minoverlap or more are considered to be overlapping.
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 or RangedData: See also findOverlapsOfPeaks. Or any parameter need to be passed into read.delim function for 2RangedData function.
header	TRUE or FALSE, default to FALSE, indicates whether data file has header
data.BED	BED format data frame or BED filename, please refer to http://genome.ucsc.edu/FAQ/FAQformat#for for details
data.GFF	GFF format data frame or GFF file name, please refer to http://genome.ucsc.edu/FAQ/FAQformat#for for details

Details

findOverlappingPeaks is now deprecated wrappers for findOverlapsOfPeaks

See Also

Deprecated, findOverlapsOfPeaks, toGRanges

condenseMatrixByColnames *Condense matrix by colnames*

Description

Condense matrix by colnames

Usage

```
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

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

```
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

```
ensemblIDs = c("ENSG00000115956", "ENSG00000071082", "ENSG00000071054",
 "ENSG00000115594", "ENSG00000115594", "ENSG00000115598", "ENSG00000170417")
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

```
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

Normalization, bias correction, and peak calling for ChIP-seq Aaron Diaz, Kiyoub Park, Daniel A. Lim, Jun S. Song Stat Appl Genet Mol Biol. Author manuscript; available in PMC 2012 May 3.Published in final edited form as: Stat Appl Genet Mol Biol. 2012 Mar 31; 11(3): 10.1515/1544-6115.1750 /j/sagmb.2012.11.issue-3/1544-6115.1750/1544-6115.1750.xml. Published online 2012 Mar 31. doi: 10.1515/1544-6115.1750 PMCID: PMC3342857

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)
```

---

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

```
data(enrichedGO)
```

Format

A list of 3 dataframes.

`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

`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

`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

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

---

estFragmentLength *estimate the fragment length*

Description

estimate the fragment length for bam files

Usage

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

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
...	Not used.

Value

numberic vector

Author(s)

Jianhong Ou

Examples

```
if(interactive()){
    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

numberic vector

Author(s)

Jianhong Ou

Examples

```
if(interactive()){
    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))
    }
}
```

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

Description

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

Usage

```
data(ExonPlusUtr.human.GRCh37)
```

Format

RangedData 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.

`strand` 1 for positive strand and -1 for negative strand

`description` description of the transcript

`ensembl_gene_id` gene id

`utr5start` 5' UTR start

`utr5end` 5' UTR end

`utr3start` 3' UTR start

`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, featureType="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 dwonstream from the feature.gr.
`zeroAt`	zero point position of feature.gr
`n.tile`	The number of tiles to generate for each element of feature.gr, default is 100
`...`	any paramters could be used by matplot

Value

invisible matrix of the plot.

Author(s)

Jianhong Ou

See Also

See Also as featureAlignedSignal, featureAlignedHeatmap

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")
```

---

featureAlignedExtendSignal

extract signals in given ranges from bam files

Description

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

Usage

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

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 dwonstream 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 numberic vector with length 1. Adjust the fragments/reads length to.
...	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

```
if(interactive()){
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 dwonstream 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 of 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, ...)
```

Arguments

`cvglists`	List of SimpleRleList or RleList
`feature.gr`	An object of GRanges with identical width.
`upstream, downstream`	Set the feature.gr to upstream and dwonstream 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.

Author(s)

Jianhong Ou

See Also

See Also as featureAlignedHeatmap, 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))
featureAlignedSignal(cvglists, feature.gr)
```

`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.

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.

`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 = "nearestBiDirectionalPromoters" 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 sec ond one must be no less than 1. For details, see bindingType.

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

`...` Not used.

Value

Output is a GRanges object of the annotated peaks.

Author(s)

Jianhong Ou

See Also

See Also as annotatePeakInBatch

Examples

```
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)
```

---

findOverlappingPeaks *Find the overlapping peaks for two peak ranges.*

Description

Find the overlapping peaks for two input peak ranges.

This function is to keep the backward compatibility with previous versions for RangedData object.

The new function findOverlapsOfPeaks is recommended.

Convert RangedData to GRanges with toGRanges function.

Usage

```
findOverlappingPeaks(Peaks1, Peaks2, maxgap = 0L,
    minoverlap=1L, 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	RangedData: See example below.
Peaks2	RangedData: See example below.
maxgap	Non-negative integer. Intervals with a separation of maxgap or less are considered to be overlapping.
minoverlap	Non-negative integer. Intervals with an overlapping of minoverlap or more are considered to be overlapping.

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 or RangedData: See also findOverlapsOfPeaks.

Details

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	RangedData contains merged overlapping peaks

Author(s)

Lihua Julie Zhu

References

1.Interval tree algorithm from: Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford. Introduction to Algorithms, second edition, MIT Press and McGraw-Hill. ISBN 0-262-53196-8

2.Zhu L.J. et al. (2010) ChIPpeakAnno: a Bioconductor package to annotate ChIP-seq and ChIP-chip data. BMC Bioinformatics 2010, 11:237 doi:10.1186/1471-2105-11-237

3. Zhu L (2013). Integrative analysis of ChIP-chip and ChIP-seq dataset. In Lee T and Luk ACS (eds.), Tilling Arrays, volume 1067, chapter 4, pp. -19. Humana Press. http://dx.doi.org/10.1007/978-1-62703-607-8_8

See Also

findOverlapsOfPeaks, annotatePeakInBatch, makeVennDiagram

Examples

```
if (interactive())
{
peaks1 =
    RangedData(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),space=c(6,6,6,6,5))
peaks2 =
    RangedData(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),space=c(6,6,6,6,5))
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=0L, minoverlap=1L,
                ignore.strand=TRUE, connectedPeaks=c("min", "merge", "keepAll"))
```

Arguments

`...`	Objects of GRanges: See example below.
`maxgap`	Non-negative integer. Peak intervals with a separation of maxgap or less are considered to be overlapped.
`minoverlap`	Non-negative integer. Peak intervals with an overlapping of minoverlap or more are considered to be overlapped.
`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 1, while set it to "min" will count it as the minimal involved peaks in any group of connected/overlapped peaks.

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 formated rownames.

Author(s)

Jianhong Ou

References

1.Interval tree algorithm from: Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford. Introduction to Algorithms, second edition, MIT Press and McGraw-Hill. ISBN 0-262-53196-8

2.Zhu L.J. et al. (2010) ChIPpeakAnno: a Bioconductor package to annotate ChIP-seq and ChIP-chip data. BMC Bioinformatics 2010, 11:237doi:10.1186/1471-2105-11-237

3. Zhu L (2013). "Integrative analysis of ChIP-chip and ChIP-seq dataset." In Lee T and Luk ACS (eds.), Tilling Arrays, volume 1067, chapter 4, pp. -19. Humana Press. http://dx.doi.org/10.1007/978-1-62703-607-8_8, http://link.springer.com/protocol/10.1007%2F978-1-62703-607-8_8

See Also

annotatePeakInBatch, makeVennDiagram, getVennCounts, findOverlappingPeaks

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="+")
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="+")
t1 <- findOverlapsOfPeaks(peaks1, peaks2, maxgap=1000)
makeVennDiagram(t1)
t1$venn_cnt
t1$peaklist
```

`findVennCounts` *Obtain Venn Counts for Venn Diagram, internal function for makeVennDigram*

Description

Obtain Venn Counts for two peak ranges using chromosome ranges or feature field, internal function for makeVennDigram

Usage

```
findVennCounts(Peaks, NameOfPeaks, maxgap = 0L, minoverlap = 1L,
               totalTest, useFeature=FALSE)
```

Arguments

`Peaks`	RangedDataList: 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`	Non-negative integer. Intervals with a separation of maxgap or less are considered to be overlapping.
`minoverlap`	Non-negative integer. Intervals with an overlapping of minoverlap or more are considered to be overlapping.
`totalTest`	Numeric value to specify the total number of tests performed to obtain the list of peaks.
`useFeature`	TRUE or FALSE, default FALSE, true means using feature field in the RangedData for calculating overlap, false means using chromosome range for calculating overlap.

Value

`p.value`	hypergeometric testing result
`vennCounts`	vennCounts objects containing counts for Venn Diagram generation, see details in limma package vennCounts

Author(s)

Lihua Julie Zhu

See Also

makeVennDiagram

getAllPeakSequence *Obtain genomic sequences around the peaks*

Description

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

Usage

```
getAllPeakSequence(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
genome	BSgenome object or mart object. Please refer to available.genomes in BSgenome package and useMart in bioMaRt package for details
AnnotationData	RangedData used if mart object is parsed in which can be obtained from getAnnotation with featureType="TSS". For example, data(TSS.human.NCBI36), data(TSS.mouse.NCBIM data(GO.rat.RGSC3.4) and data(TSS.zebrafish.Zv8). If not supplied, then annotation will be obtained from biomaRt automatically using the mart object

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

Durinck S. et al. (2005) BioMart and Bioconductor: a powerful link between biological biomarts and microarray data analysis. Bioinformatics, 21, 3439-3440.

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)
seq <- 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 or RangedData 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.

`strand`	1 for positive strand and -1 for negative strand where the feature is located
`description`	description of the feeature 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

Author(s)

Lihua Julie Zhu, Jianhong Ou

References

Durinck S. et al. (2005) BioMart and Bioconductor: a powerful link between biological biomarts and microarray data analysis. Bioinformatics, 21, 3439-3440.

Examples

```
if (interactive())
{
  mart <- useMart(biomart="ensembl", dataset="hsapiens_gene_ensembl")
  Annotation <- getAnnotation(mart, featureType="TSS")
}
```

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.eg 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,
              minGOterm=10, multiAdjMethod=NULL,
              condense=FALSE,
              removeAncestorByPval=NULL,
              keepByLevel=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.

Value

A list with 3 elements

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

`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

`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

References

Johnson, N. L., Kotz, S., and Kemp, A. W. (1992) Univariate Discrete Distributions, Second Edition. New York: Wiley

See Also

phyper, hyperGtest

Examples

```
data(enrichedGO)
enrichedGO$mf[1:10,]
enrichedGO$bp[1:10,]
enrichedGO$cc
if (interactive()) {
   data(annotatedPeak)
   library(org.Hs.eg.db)
   enriched.GO = getEnrichedGO(annotatedPeak[1:6,],
                               orgAnn="org.Hs.eg.db",
```

```
                                maxP=0.01,
                                minGOterm=10,
                                multiAdjMethod= NULL)
    dim(enriched.GO$mf)
    colnames(enriched.GO$mf)
    dim(enriched.GO$bp)
    enriched.GO$cc
}
```

`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 path 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)
```

Arguments

`annotatedPeak`	GRanges such as data(annotatedPeak) 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
`pathAnn`	pathway annotation package such as KEGG.db, reactome.db
`feature_id_type`	the feature type in annotatedPeakRanges such as ensembl_gene_id, refseq_id, gene_symbol or entrez_id
`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

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

References

Johnson, N. L., Kotz, S., and Kemp, A. W. (1992) Univariate Discrete Distributions, Second Edition. New York: Wiley

See Also

phyper, hyperGtest

Examples

```
if (interactive()) {
data(annotatedPeak)
library(org.Hs.eg.db)
library(reactome.db)
enriched.PATH = getEnrichedPATH(annotatedPeak, orgAnn="org.Hs.eg.db",
                 pathAnn="reactome.db", maxP=0.01,
                 minPATHterm=10, multiAdjMethod=NULL)
 head(enriched.PATH)
}
```

---

getVennCounts *Obtain Venn Counts for Venn Diagram, internal function for makeVennDigram*

Description

Obtain Venn Counts for peak ranges using chromosome ranges or feature field, internal function for makeVennDigram

Usage

```
getVennCounts(..., maxgap = 0L, minoverlap=1L,
    by=c("region", "feature", "base"),
    ignore.strand=TRUE, connectedPeaks=c("min", "merge", "keepAll"))
```

Arguments

...
: Objects of GRanges or RangedData: See example below.

maxgap
: Non-negative integer. Intervals with a separation of maxgap or less are considered to be overlapping.

minoverlap
: Non-negative integer. Intervals with an overlapping of minoverlap or more are considered to be overlapping.

by	region, feature or base, default region. feature means using feature field in the RangedData or 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 concered 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()){
peaks1 = RangedData(IRanges(start = c(967654, 2010897, 2496704),
                            end = c(967754, 2010997, 2496804),
                            names = c("Site1", "Site2", "Site3")),
                    space = c("1", "2", "3"),
                    strand=as.integer(1),
                    feature=c("a","b", "c"))
  peaks2 =
      RangedData(IRanges(start=c(967659, 2010898, 2496700, 3075866, 3123260),
                         end=c(967869, 2011108, 2496920, 3076166, 3123470),
                         names = c("t1", "t2", "t3", "t4", "t5")),
                  space = c("1", "2", "3", "1", "2"),
                  strand = c(1, 1, -1,-1,1),
                  feature=c("a","c","d","e", "a"))
    getVennCounts(peaks1,peaks2, maxgap=0)
    getVennCounts(peaks1,peaks2, maxgap=0, by="feature")
    getVennCounts(peaks1, peaks2, maxgap=0, by="base")
}
```

GFF2RangedData *Convert GFF format to RangedData*

Description

Convert GFF format to RangedData. This function will be depreciated. Use function toGRanges instead.

Usage

```
GFF2RangedData(data.GFF,header=FALSE, ...)
```

Arguments

data.GFF	GFF format data frame or GFF file name, please refer to http://genome.ucsc.edu/FAQ/FAQformat#for for details
header	TRUE or FALSE, default to FALSE, indicates whether data.GFF file has GFF header
...	any parameter need to be passed into read.delim function

Value

RangedData 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.

strand	1 for positive strand and -1 for negative strand where the feature is located.

Note

For converting the peakList in GFF format to RangedData before calling annotatePeakInBatch function

Author(s)

Lihua Julie Zhu

Examples

```
test.GFF = data.frame(cbind(seqname  = c("chr1", "chr2"),
source=rep("Macs", 2),
feature=rep("peak", 2),
start=c("100", "1000"),
end=c("200", "1100"),
score=c(60, 26),
strand=c(1, -1),
frame=c(".", 2),
group=c("peak1", "peak2")))
test.rangedData = GFF2RangedData(test.GFF)
```

HOT.spots *High Occupancy of Transcription Related Factors regions*

Description

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

Usage

```
data("HOT.spots")
```

Format

An object of GRangesList

Details

How to generated the data:

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

Yip KY, Cheng C, Bhardwaj N, Brown JB, Leng J, Kundaje A, Rozowsky J, Birney E, Bickel P, Snyder M, Gerstein M. Classification of human genomic regions based on experimentally determined binding sites of more than 100 transcription-related factors. Genome Biol. 2012 Sep 26;13(9):R48. doi: 10.1186/gb-2012-13-9-r48. PubMed PMID: 22950945; PubMed Central PMCID: PMC3491392.

Examples

```
data(HOT.spots)
elementNROWS(HOT.spots)
```

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

Usage

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

Arguments

peaksA, peaksB	peaklist, GRanges object.
bamfileA, bamfileB	file path of bam files.
maxgap	Non-negative integer. Peak intervals with a separation of maxgap or less are considered to be overlapped.
minoverlap	Non-negative integer. Peak intervals with an overlapping of minoverlap or more are considered to be overlapped.
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

Li, Qunhua, et al. "Measuring reproducibility of high-throughput experiments." The annals of applied statistics (2011): 1752-1779.

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")
```

```
            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 = 0L, minoverlap = 1L,
                totalTest, by = c("region", "feature", "base"),
                ignore.strand = TRUE, connectedPeaks = c("min",
                "merge", "keepAll"), method = c("hyperG",
                "permutation"), TxDb, ...)
```

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`	Non-negative integer. Intervals with a separation of maxgap or less are considered to be overlapping.
`minoverlap`	Non-negative integer. Intervals with an overlapping of minoverlap or more are considered to be overlapping.
`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.
`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.
`method`	method used for p value calculation. hyperG means hypergeometric test and permutation means peakPermTest
`TxDb`	An object of TxDb
`...`	Additional arguments to be passed to venn.diagram

Details

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

Value

In addition to a Venn Diagram produced, a p.value is calculated by hypergeometric test to determine whether the peaks or features are overlapped significantly.

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,
                                          3075866,3123260),
                                end = c(967869, 2011108, 2496920,
                                        3076166, 3123470),
                                names = c("t1", "t2", "t3", "t4", "t5")),
                        strand = c("+", "+", "-", "-", "+"),
                        feature=c("a","b","c","d","a"))
    makeVennDiagram(list(peaks1, peaks2), NameOfPeaks=c("TF1", "TF2"),
                    totalTest=100,scaled=FALSE, euler.d=FALSE)

    makeVennDiagram(list(peaks1, peaks2), NameOfPeaks=c("TF1", "TF2"),
                    totalTest=100)

    ###### 4-way diagram using annotated feature instead of chromosome ranges

    makeVennDiagram(list(peaks1, peaks2, peaks1, peaks2),
                    NameOfPeaks=c("TF1", "TF2","TF3", "TF4"),
                    totalTest=100, by="feature",
                    main = "Venn Diagram for 4 peak lists",
                    fill=c(1,2,3,4))
}
```

---

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
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 nagative 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

Zhu L.J. et al. (2010) ChIPpeakAnno: a Bioconductor package to annotate ChIP-seq and ChIP-chip data. BMC Bioinformatics 2010, 11:237doi:10.1186/1471-2105-11-237

See Also

annotatePeakInBatch, findOverlappingPeaks, makeVennDiagram

Examples

```
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")
}
```

`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

```
data(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

Robertson G, Hirst M, Bainbridge M, Bilenky M, Zhao Y, et al. (2007) Genome-wide profiles of 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, last = 1e+06)
```

Arguments

`sequence`	The sequences packaged in DNAStringSet, DNAString object or output of function getAllPeakSequence.
`MarkovOrder`	Markov order.
`last`	The sequence size to be analyzed.

Value

A numeric vector.

Author(s)

Jianhong Ou

See Also

See Also as oligoSummary

Examples

```
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 Markove chain.

Usage

```
oligoSummary(sequence, oligoLength = 6L, freqs = NULL,
             MarkovOrder = 3L, quickMotif = FALSE, revcomp=FALSE,
             maxsize=100000)
```

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).

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

van Helden, Jacques, Marcel li del Olmo, and Jose E. Perez-Ortin. "Statistical analysis of yeast genomic downstream sequences reveals putative polyadenylation signals." Nucleic Acids Research 28.4 (2000): 1000-1010.

See Also

See Also as frequency

Examples

```
if(interactive()){
    data(annotatedPeak)
    library(BSgenome.Hsapiens.UCSC.hg19)
    seq <- getAllPeakSequence(annotatedPeak[1:100],
                 upstream=20,
                 downstream=20,
                 genome=Hsapiens)
    oligoSummary(seq)
}
```

---

peakPermTest *Permutation Test for two given peak lists*

Description

Performs a permutation test to seee 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=0L, 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

Davison, A. C. and Hinkley, D. V. (1997) Bootstrap methods and their application, Cambridge University Press, United Kingdom, 156-160

See Also

preparePool, bindist

Examples

```
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

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

Format

RangedData 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 space containing the chromosome where the peak is located.

References

Philippe Lefranois, Ghia M Euskirchen, Raymond K Auerbach, Joel Rozowsky, Theodore Gibson, Christopher M Yellman, Mark Gerstein and Michael Snyder (2009) Efficient yeast ChIP-Seq using multiplex short-read DNA sequencing BMC Genomics 10:37

Examples

```
data(Peaks.Ste12.Replicate1)
str(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

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

Format

RangedData 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 space containing the chromosome where the peak is located.

Source

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

References

Philippe Lefranois, Ghia M Euskirchen, Raymond K Auerbach, Joel Rozowsky, Theodore Gibson, Christopher M Yellman, Mark Gerstein and Michael Snyder (2009) Efficient yeast ChIP-Seq using multiplex short-read DNA sequencing BMC Genomics 10:37doi:10.1186/1471-2164-10-37

Examples

```
data(Peaks.Ste12.Replicate2)
str(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

```
data(Peaks.Ste12.Replicate3)
```

Format

RangedData 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 space containing the chromosome where the peak is located.

Source

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

References

Philippe Lefranois, Ghia M Euskirchen, Raymond K Auerbach, Joel Rozowsky, Theodore Gibson, Christopher M Yellman, Mark Gerstein and Michael Snyder (2009) Efficient yeast ChIP-Seq using multiplex short-read DNA sequencing BMC Genomics 10:37doi:10.1186/1471-2164-10-37

Examples

```
data(Peaks.Ste12.Replicate3)
str(Peaks.Ste12.Replicate3)
```

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 or RangedData: 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

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
percentPeaksWithBDP	The percent of input peaks containing bi-directional promoters
n.peaks	The total number of input peaks
n.peaksWithBDP	The # of input peaks containing bi-directional promoters

Author(s)

Lihua Julie Zhu, Jianhong Ou

References

Zhu L.J. et al. (2010) ChIPpeakAnno: a Bioconductor package to annotate ChIP-seq and ChIP-chip data. BMC Bioinformatics 2010, 11:237doi:10.1186/1471-2105-11-237

See Also

annotatePeakInBatch, findOverlappingPeaks, makeVennDiagram

Examples

```
if (interactive())
{
    data(myPeakList)
    data(TSS.human.NCBI36)
    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.

Objects from the Class

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

Slots

grs object of "GRangesList" The list of binding ranges

N vector of "integer", permutation number for each ranges

Methods

$, $<- Get or set the slot of permPool

See Also

preparePool, peakPermTest

pie1 *Pie Charts*

Description

Draw a pie chart with percentage

Usage

```
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 lable, 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)
```

---

preparePool *prepare data for permutation test*

Description

prepare data for permutation test peakPermTest

Usage

```
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

Examples

```
if(interactive()){
    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

```
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

```
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

```
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, alia 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

```
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 of each pattern in the sequences.*

Description

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

Usage

```
summarizePatternInPeaks(patternFilePath, format = "fasta",skip=0L,
                        BSgenomeName, peaks, outfile, append = FALSE)
```

Arguments

patternFilePath	A character vector containing the path to the file to read the patterns from.
format	Either "fasta" (the default) or "fastq"
skip	Single non-negative integer. The number of records of the pattern file to skip before beginning to read in records.
BSgenomeName	BSgenome object. Please refer to available.genomes in BSgenome package for details
peaks	GRanges or RangedData containing the peaks
outfile	A character vector containing the path to the file to write the summary output.
append	TRUE or FALSE, default FALSE

Value

A data frame with 3 columns as n.peaksWithPattern (number of peaks with the pattern), n.totalPeaks (total number of peaks in the input) and Pattern (the corresponding pattern). The summary will consider both strand (including reverse complement).

Author(s)

Lihua Julie Zhu

Examples

```
peaks = RangedData(IRanges(start=c(100, 500), end=c(300, 600),
                           names=c("peak1", "peak2")),
                   space=c("NC_008253", "NC_010468"))
filepath =system.file("extdata", "examplePattern.fa",
                      package="ChIPpeakAnno")
library(BSgenome.Ecoli.NCBI.20080805)
summarizePatternInPeaks(patternFilePath=filepath, format="fasta",
                        skip=0L, BSgenomeName=Ecoli, peaks=peaks)
```

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=1e6, step=1e6,
                      keepPartialWindow=FALSE,
                      mode=countByOverlaps, ...)
```

Arguments

reads	A 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, alia 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

```
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, ...)
```

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.
`...`	Not used.

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, 300, 900,
                             300, 500, 500),
                     names=letters[1:11]))
se <- tileGRanges(genes, windowSize=50, step=10)
```

toGRanges *Convert dataset to GRanges*

Description

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

Usage

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

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.
feature	annotation type
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 and only if the first row contains one fewer field than the number of columns.
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.
...	parameters passed to read.table
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

```
macs <- system.file("extdata", "MACS_peaks.xls", package="ChIPpeakAnno")
macsOutput <- toGRanges(macs, format="MACS")
if(interactive()){
  ## MACS connection
  macs <- readLines(macs)
  macs <- textConnection(macs)
  macsOutput <- toGRanges(macs, format="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")
  ## 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")
  macs <- read.delim(macs, comment.char="#")
  toGRanges(macs)
}
```

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-> [C|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 = "AACCNWMK"
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

```
data(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 strinad information. In addition, the following variables are included.

`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

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

Format

A 'GRanges' [package "GenomicRanges"] object with ensembl id as names.

Details

used in the examples Annotation data obtained by:

mart = useMart(biomart = "ensembl", dataset = "hsapiens_gene_ensembl")

getAnnotation(mart, featureType = "TSS")

Examples

```
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

```
data(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 strinad information. In addition, the following variables are included.

`description` description of the gene

Details

used in the examples Annotation data obtained by:

mart = useMart(biomart = "ensembl_mart_47", dataset = "hsapiens_gene_ensembl", archive=TRUE)

getAnnotation(mart, featureType = "TSS")

Examples

```
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

```
data(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 strinad information. In addition, the following variables are included.

`description` description of the gene

Details

Annotation data obtained by:

mart = useMart(biomart = "ensembl", dataset = "mmusculus_gene_ensembl")

getAnnotation(mart, featureType = "TSS")

Examples

```
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

```
data(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.

`description` description of the gene

Details

Annotation data obtained by:

mart = useMart(biomart = "ensembl", dataset = "mmusculus_gene_ensembl")

getAnnotation(mart, featureType = "TSS")

Examples

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

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

Description

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

Usage

```
data(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.

`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.RGSC3.4)
slotNames(TSS.rat.RGSC3.4)
```

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

Description

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

Usage

```
data(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.

`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` *TSS annotation data for zebrafish (Zv8) obtained from biomaRt*

Description

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

Usage

```
data(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.

`description` description of the gene

Details

Annotation data obtained by: mart <- useMart(biomart="ENSEMBL_MART_ENSEMBL", host="may2009.archive.ense
path="/biomart/martservice", dataset="drerio_gene_ensembl")

getAnnotation(mart, featureType = "TSS")

Examples

```
data(TSS.zebrafish.Zv8)
slotNames(TSS.zebrafish.Zv8)
```

TSS.zebrafish.Zv9 *TSS annotation for Danio rerio (Zv9) obtained from biomaRt*

Description

TSS annotation for Danio rerio (Zv9) obtained from biomaRt

Usage

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

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.

`description` description of the gene

Details

Annotation data obtained by:

mart <- useMart(biomart="ENSEMBL_MART_ENSEMBL", host="mar2015.archive.ensembl.org", path="/biomart/martservice", dataset="drerio_gene_ensembl")

getAnnotation(mart, featureType = "TSS")

Examples

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

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

```
data("wgEncodeTfbsV3")
```

Format

An object of GRanges.

Details

How to generate the data:

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/wgEncodeRegTfbsClustere

Examples

```
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 writeFASTA 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 varibles 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("CCCCCCCCGGGGG", "TTTTTTTAAAAAA"))

write2FASTA(peaksWithSequences, file="testseq.fasta", width=50)
```

---

xget *Return the value from a Bimap objects*

Description

Search by name for an Bimap object.

Usage

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

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