Package 'ORFik'

November 18, 2019

Type Package

Title Open Reading Frames in Genomics

Version 1.6.2

Encoding UTF-8

Description Tools for manipulation of sequence-, RiboSeq-, RNASeq- and CageSeq data. ORFik is extremely fast through use of C, data.table and GenomicRanges. Package allows to reassign starts of the transcripts with the use of CageSeq data, automatic shifting of RiboSeq reads, finding of Open Reading Frames for whole genomes and much more.

biocViews ImmunoOncology, Software, Sequencing, RiboSeq, RNASeq, FunctionalGenomics, Coverage, Alignment, DataImport

License MIT + file LICENSE

LazyData TRUE

BugReports https://github.com/Roleren/ORFik/issues

URL https://github.com/Roleren/ORFik

Depends R (>= 3.6.0), IRanges (>= 2.17.1), GenomicRanges (>= 1.35.1), GenomicAlignments (>= 1.19.0)

Imports S4Vectors (>= 0.21.3), GenomeInfoDb (>= 1.15.5), GenomicFeatures (>= 1.31.10), AnnotationDbi (>= 1.45.0), rtracklayer (>= 1.43.0), Rcpp (>= 1.0.0), data.table (>= 1.11.8), Biostrings (>= 2.51.1), stats, tools, Rsamtools (>= 1.35.0), BiocGenerics (>= 0.29.1), ggplot2 (>= 2.2.1), gridExtra (>= 2.3), methods (>= 3.6.0)

RoxygenNote 6.1.1

Suggests testthat, rmarkdown, knitr, BiocStyle, BSgenome, BSgenome.Hsapiens.UCSC.hg19

LinkingTo Rcpp

VignetteBuilder knitr

git_url https://git.bioconductor.org/packages/ORFik

git_branch RELEASE_3_10

git_last_commit 9094c9c

git_last_commit_date 2019-11-14

Date/Publication 2019-11-17

Author Haakon Tjeldnes [aut, cre, dtc],
Kornel Labun [aut, cph],
Katarzyna Chyzynska [ctb, dtc],
Evind Valen [ths, fnd]

Maintainer Haakon Tjeldnes <hauken_heyken@hotmail.com>

R topics documented:

`ORFik-package` *ORFik for analysis of open reading frames.*

Description

Main goals:

1. Finding Open Reading Frames (very fast) in the genome of interest or on the set of transcripts/sequences.
2. Utilities for metaplots of RiboSeq coverage over gene START and STOP codons allowing to spot the shift.
3. Shifting functions for the RiboSeq data.
4. Finding new Transcription Start Sites with the use of CageSeq data.
5. Various measurements of gene identity e.g. FLOSS, coverage, ORFscore, entropy that are recreated based on many scientific publications.
6. Utility functions to extend GenomicRanges for faster grouping, splitting, tiling etc.

Author(s)

Maintainer: Haakon Tjeldnes <hauken_heyken@hotmail.com> [data contributor]

Authors:

- Kornel Labun <kornellabun@gmail.com> [copyright holder]

Other contributors:

- Katarzyna Chyzynska <katchyz@gmail.com> [contributor, data contributor]
- Evind Valen <eivind.valen@gmail.com> [thesis advisor, funder]

See Also

Useful links:

- https://github.com/Roleren/ORFik
- Report bugs at https://github.com/Roleren/ORFik/issues

`addCdsOnLeaderEnds` *Extends leaders downstream*

Description

When finding uORFs, often you want to allow them to end inside the cds.

Usage

```
addCdsOnLeaderEnds(fiveUTRs, cds, onlyFirstExon = FALSE)
```

Arguments

`fiveUTRs`	The 5' leader sequences as GRangesList
`cds`	If you want to extend 5' leaders downstream, to catch uorfs going into cds, include it.
`onlyFirstExon`	logical (F), include whole cds or only first exons.

Details

This is a simple way to do that

Value

a GRangesList of cds exons added to ends

See Also

Other uorfs: `filterUORFs`, `removeORFsWithSameStartAsCDS`, `removeORFsWithSameStopAsCDS`, `removeORFsWithStartInsideCDS`, `removeORFsWithinCDS`, `uORFSearchSpace`

`addNewTSSOnLeaders` *add cage max peaks as new transcript start sites for each 5' leader (*) strands are not supported, since direction must be known.*

Description

add cage max peaks as new transcript start sites for each 5' leader (*) strands are not supported, since direction must be known.

Usage

```
addNewTSSOnLeaders(fiveUTRs, maxPeakPosition, removeUnused, cageMcol)
```

Arguments

`fiveUTRs`	(GRangesList) The 5' leaders or full transcript sequences
`maxPeakPosition`	The max peak for each 5' leader found by cage
`removeUnused`	logical (FALSE), if False: (standard is to set them to original annotation), If TRUE: remove leaders that did not have any cage support.
`cageMcol`	a logical (FALSE), if TRUE, add a meta column to the returned object with the raw CAGE counts in support for new TSS.

Value

a GRanges object of first exons

`allFeaturesHelper` *Calculate the features in computeFeatures*

Description

Not used directly, calculates all features.

Usage

```
allFeaturesHelper(grl, RFP, RNA, tx, fiveUTRs, cds, threeUTRs, faFile,
  riboStart, riboStop, orfFeatures, includeNonVarying, grl.is.sorted)
```

Arguments

`grl`	a `GRangesList` object with usually ORFs, but can also be either leaders, cds', 3' utrs, etc.
`RFP`	RiboSeq reads as GAlignment, GRanges or GRangesList object
`RNA`	RnaSeq reads as GAlignment, GRanges or GRangesList object
`tx`	a GrangesList of transcripts, normally called from: exonsBy(Gtf, by = "tx", use.names = T) only add this if you are not including Gtf file You do not need to reassign these to the cage peaks, it will do it for you.

fiveUTRs	fiveUTRs as GRangesList, if you used cage-data to extend 5' utrs, remember to input CAGE assigned version and not original!
cds	a GRangesList of coding sequences
threeUTRs	a GrangesList of transcript 3' utrs, normally called from: threeUTRsByTranscript(Gtf, use.names = T)
faFile	a FaFile or BSgenome from the fasta file, see ?FaFile
riboStart	usually 26, the start of the floss interval, see ?floss
riboStop	usually 34, the end of the floss interval
orfFeatures	a logical, is the grl a list of orfs?
includeNonVarying	a logical, if TRUE, include all features not dependent on RiboSeq data and RNASeq data, that is: Kozak, fractionLengths, distORFCDS, isInFrame, isOverlapping and rankInTx
grl.is.sorted	logical (F), a speed up if you know argument grl is sorted, set this to TRUE.

Value

a data.table with features

`assignAnnotations` *Overlaps GRanges object with provided annotations.*

Description

It will return same list of GRanges, but with metdata columns: trainscript_id - id of transcripts that overlap with each ORF gene_id - id of gene that this transcript belongs to isoform - for coding protein alignment in relation to cds on coresponding transcript, for non-coding transcripts alignment in relation to the transcript.

Usage

```
assignAnnotations(ORFs, con)
```

Arguments

ORFs	- GRanges or GRangesList object of your ORFs.
con	- Path to gtf file with annotations.

Value

A GRanges object of your ORFs with metadata columns 'gene', 'transcript', isoform' and 'biotype'.

assignFirstExonsStartSite

Reassign the start positions of the first exons per group in grl

Description

make sure your grl is sorted, since start of "-" strand objects should be the max end in group, use ORFik:::sortPerGroup(grl) to get sorted grl.

Usage

```
assignFirstExonsStartSite(grl, newStarts)
```

Arguments

grl	a GRangesList object
newStarts	an integer vector of same length as grl, with new start values

Value

the same GRangesList with new start sites

See Also

Other GRanges: assignLastExonsStopSite, downstreamFromPerGroup, downstreamOfPerGroup, upstreamFromPerGroup, upstreamOfPerGroup

assignLastExonsStopSite

Reassign the stop positions of the last exons per group

Description

make sure your grl is sorted, since stop of "-" strand objects should be the min start in group, use ORFik:::sortPerGroup(grl) to get sorted grl.

Usage

```
assignLastExonsStopSite(grl, newStops)
```

Arguments

grl	a GRangesList object
newStops	an integer vector of same length as grl, with new start values

Value

the same GRangesList with new stop sites

See Also

Other GRanges: assignFirstExonsStartSite, downstreamFromPerGroup, downstreamOfPerGroup, upstreamFromPerGroup, upstreamOfPerGroup

---

assignTSSByCage *Input a txdb and add a 5' leader for each transcript, that does not have one.*

Description

For all cds in txdb, that does not have a 5' leader: Start at 1 base upstream of cds and use CAGE, to assign leader start. All these leaders will be 1 exon based, if you really want exon splicings, you can use exon prediction tools, or run sequencing experiments.

Usage

```
assignTSSByCage(txdb, cage, extension = 1000, filterValue = 1,
  restrictUpstreamToTx = FALSE, removeUnused = FALSE,
  preCleanup = TRUE)
```

Arguments

txdb	a TxDb file, an ORFik experiment or a path to one of: (.gtf ,.gff, .gff2, .gff2, .db or .sqlite)
cage	Either a filePath for the CageSeq file as .bed .bam or .wig, with possible compressions (".gzip", ".gz", ".bgz"), or already loaded CageSeq peak data as GRanges or GAlignment. NOTE: If it is a .bam file, it will add a score column by running: convertToOneBasedRanges(cage, method = "5prime", addScoreColumn = TRUE)
extension	The maximum number of basses upstream of the TSS to search for CageSeq peak.
filterValue	The minimum number of reads on cage position, for it to be counted as possible new tss. (represented in score column in CageSeq data) If you already filtered, set it to 0.
restrictUpstreamToTx	a logical (FALSE). If TRUE: restrict leaders to not extend closer than 5 bases from closest upstream leader, set this to TRUE.
removeUnused	logical (FALSE), if False: (standard is to set them to original annotation), If TRUE: remove leaders that did not have any cage support.
preCleanup	logical (TRUE), if TRUE, remove all reads in region (-5:-1, 1:5) of all original tss in leaders. This is to keep original TSS if it is only +/- 5 bases from the original.

Details

Given a TxDb object, reassign the start site per transcript using max peaks from CageSeq data. A max peak is defined as new TSS if it is within boundary of 5' leader range, specified by 'extension' in bp. A max peak must also be higher than minimum CageSeq peak cutoff specified in 'filterValue'. The new TSS will then be the positioned where the cage read (with highest read count in the interval).

Value

a TxDb obect of reassigned transcripts

See Also

Other CAGE: reassignTSSbyCage, reassignTxDbByCage

Examples

```
## Not run:
library(GenomicFeatures)
# Get the gtf txdb file
txdbFile <- system.file("extdata", "hg19_knownGene_sample.sqlite",
package = "GenomicFeatures")
cagePath <- system.file("extdata", "cage-seq-heart.bed.bgz",
package = "ORFik")
reassignTxDbByCage(txdbFile, cagePath)

## End(Not run)
```

---

asTX *Map genomic to transcript coordinates by reference*

Description

Similar to GenomicFeatures' pmapToTranscripts, but in this version the grl ranges are compared to reference ranges with same name, not by index. And it has a security fix.

Usage

```
asTX(grl, reference)
```

Arguments

grl	a GRangesList of ranges within the reference, grl must have column called names that gives grouping for result
reference	a GrangesList of ranges that include and are bigger or equal to grl ig. cds is grl and gene can be reference

Value

a GRangesList in transcript coordinates

See Also

Other ExtendGenomicRanges: coveragePerTiling, overlapsToCoverage, reduceKeepAttr, tile1, txSeqsFromFa, windowPerGroup

bamVarName *Get library variable names from ORFik experiment*

Description

What will each sample be called given the columns of the experiment?

Usage

```
bamVarName(df, skip.replicate = length(unique(df$rep)) == 1,
  skip.condition = length(unique(df$condition)) == 1,
  skip.stage = length(unique(df$stage)) == 1,
  skip.fraction = length(unique(df$fraction)) == 1,
  skip.experiment = !df@expInVarName)
```

Arguments

df	an ORFik experiment data.frame
skip.replicate	a logical (FALSE), don't include replicate in variable name.
skip.condition	a logical (FALSE), don't include condition in variable name.
skip.stage	a logical (FALSE), don't include stage in variable name.
skip.fraction	a logical (FALSE), don't include fraction
skip.experiment	a logical (FALSE), don't include experiment

Value

variable names of libraries (character vector)

bamVarNamePicker *Get variable name per filepath in experiment*

Description

Get variable name per filepath in experiment

Usage

```
bamVarNamePicker(df, skip.replicate = FALSE, skip.condition = FALSE,
  skip.stage = FALSE, skip.fraction = FALSE, skip.experiment = FALSE)
```

Arguments

df	an ORFik experiment data.frame
skip.replicate	a logical (FALSE), don't include replicate in variable name.
skip.condition	a logical (FALSE), don't include condition in variable name.
skip.stage	a logical (FALSE), don't include stage in variable name.
skip.fraction	a logical (FALSE), don't include fraction
skip.experiment	a logical (FALSE), don't include experiment

Value

variable name of library (character vector)

`bedToGR` *Converts different type of files to Granges*

Description

column 5 will be set to score Only Accepts bed files for now, standard format from Fantom5

Usage

```
bedToGR(x, bed6 = TRUE)
```

Arguments

`x`	An data.frame from imported bed-file, to convert to GRanges
`bed6`	If bed6, no meta column is added

Value

a GRanges object from bed

See Also

Other utils: `convertToOneBasedRanges`, `findFa`, `fread.bed`, `optimizeReads`, `readBam`, `readWig`

`changePointAnalysis` *Get the offset for specific RiboSeq read width*

Description

Get the offset for specific RiboSeq read width

Usage

```
changePointAnalysis(x, feature = "start")
```

Arguments

`x`	a vector with count per position to analyse, assumes the zero is in the middle + 1 (position 0)
`feature`	(character) either "start" or "stop"

Value

a single numeric offset

See Also

Other pshifting: `detectRibosomeShifts`, `shiftFootprints`

checkRFP *Helper Function to check valid RFP input*

Description

Helper Function to check valid RFP input

Usage

```
checkRFP(class)
```

Arguments

class,	the given class of RFP object

Value

NULL, stop if invalid object

See Also

Other validity: checkRNA, is.ORF, is.gr_or_grl, is.grl, validGRL, validSeqlevels

checkRNA *Helper Function to check valid RNA input*

Description

Helper Function to check valid RNA input

Usage

```
checkRNA(class)
```

Arguments

class,	the given class of RNA object

Value

NULL, stop if unvalid object

See Also

Other validity: checkRFP, is.ORF, is.gr_or_grl, is.grl, validGRL, validSeqlevels

codonSumsPerGroup *Get read hits per codon*

Description

Helper for entropy function, normally not used directly Seperate each group into tuples (abstract codons) Gives sum for each tuple within each group

Usage

```
codonSumsPerGroup(grl, reads)
```

Arguments

grl	a GRangesList
reads	a GRanges or GAlignment

Details

Example: counts c(1,0,0,1), with reg_len = 2, gives c(1,0) and c(0,1), these are summed and returned as data.table 10 bases, will give 3 codons, 1 base codons does not exist.

Value

a data.table with codon sums

computeFeatures *Get all possible features in ORFik*

Description

If you want to get all the features easily, you can use this function. Each feature have a link to an article describing its creation and idea behind it. Look at the functions in the feature family to see all of them.

Usage

```
computeFeatures(grl, RFP, RNA = NULL, Gtf, faFile = NULL,
  riboStart = 26, riboStop = 34, orfFeatures = TRUE,
  includeNonVarying = TRUE, grl.is.sorted = FALSE)
```

Arguments

grl	a GRangesList object with usually ORFs, but can also be either leaders, cds', 3' utrs, etc.
RFP	RiboSeq reads as GAlignment, GRanges or GRangesList object
RNA	RnaSeq reads as GAlignment, GRanges or GRangesList object
Gtf	a TxDb object of a gtf file or path to gtf, gff .sqlite etc.

faFile	a FaFile or BSgenome from the fasta file, see ?FaFile
riboStart	usually 26, the start of the floss interval, see ?floss
riboStop	usually 34, the end of the floss interval
orfFeatures	a logical, is the grl a list of orfs?
includeNonVarying	a logical, if TRUE, include all features not dependent on RiboSeq data and RNASeq data, that is: Kozak, fractionLengths, distORFCDS, isInFrame, isOverlapping and rankInTx
grl.is.sorted	logical (F), a speed up if you know argument grl is sorted, set this to TRUE.

Details

If you used CageSeq to reannotate your leaders, your txDB object must contain the reassigned leaders. Use [reassignTxDbByCage()] to get the txdb.

Value

a data.table with scores, each column is one score type, name of columns are the names of the scores, i.g [floss()] or [fpkm()]

See Also

Other features: computeFeaturesCage, disengagementScore, distToCds, distToTSS, entropy, floss, fpkm_calc, fpkm, fractionLength, initiationScore, insideOutsideORF, isInFrame, isOverlapping, kozakSequenceScore, orfScore, rankOrder, ribosomeReleaseScore, ribosomeStallingScore, startRegionCoverage, startRegion, subsetCoverage, translationalEff

Examples

```
# Here we make an example from scratch
# Usually the ORFs are found in orfik, which makes names for you etc.
gtf <- system.file("extdata", "annotations.gtf",
package = "ORFik") ## location of the gtf file
suppressWarnings(txdb <-
                   GenomicFeatures::makeTxDbFromGFF(gtf, format = "gtf"))
# use cds' as ORFs for this example
ORFs <- GenomicFeatures::cdsBy(txdb, by = "tx", use.names = TRUE)
ORFs <- makeORFNames(ORFs) # need ORF names
# make Ribo-seq data,
RFP <- unlistGrl(firstExonPerGroup(ORFs))
suppressWarnings(computeFeatures(ORFs, RFP, Gtf = txdb))
# For more details see vignettes.
```

---

computeFeaturesCage *Get all possible features in ORFik*

Description

If you have a txdb with correctly reassigned transcripts, use: [computeFeatures()]

Usage

```
computeFeaturesCage(grl, RFP, RNA = NULL, Gtf = NULL, tx = NULL,
  fiveUTRs = NULL, cds = NULL, threeUTRs = NULL, faFile = NULL,
  riboStart = 26, riboStop = 34, orfFeatures = TRUE,
  includeNonVarying = TRUE, grl.is.sorted = FALSE)
```

Arguments

grl	a GRangesList object with usually ORFs, but can also be either leaders, cds', 3' utrs, etc.
RFP	RiboSeq reads as GAlignment, GRanges or GRangesList object
RNA	RnaSeq reads as GAlignment, GRanges or GRangesList object
Gtf	a TxDb object of a gtf file or path to gtf, gff .sqlite etc.
tx	a GrangesList of transcripts, normally called from: exonsBy(Gtf, by = "tx", use.names = T) only add this if you are not including Gtf file You do not need to reassign these to the cage peaks, it will do it for you.
fiveUTRs	fiveUTRs as GRangesList, if you used cage-data to extend 5' utrs, remember to input CAGE assigned version and not original!
cds	a GRangesList of coding sequences
threeUTRs	a GrangesList of transcript 3' utrs, normally called from: threeUTRsByTranscript(Gtf, use.names = T)
faFile	a FaFile or BSgenome from the fasta file, see ?FaFile
riboStart	usually 26, the start of the floss interval, see ?floss
riboStop	usually 34, the end of the floss interval
orfFeatures	a logical, is the grl a list of orfs?
includeNonVarying	a logical, if TRUE, include all features not dependent on RiboSeq data and RNASeq data, that is: Kozak, fractionLengths, distORFCDS, isInFrame, isOverlapping and rankInTx
grl.is.sorted	logical (F), a speed up if you know argument grl is sorted, set this to TRUE.

Details

A specialized version if you don't have a correct txdb, for example with CAGE reassigned leaders while txdb is not updated. It is 2x faster for tested data. The point of this function is to give you the ability to input transcript etc directly into the function, and not load them from txdb. Each feature have a link to an article describing feature, try ?floss

Value

a data.table with scores, each column is one score type, name of columns are the names of the scores, i.g [floss()] or [fpkm()]

See Also

Other features: computeFeatures, disengagementScore, distToCds, distToTSS, entropy, floss, fpkm_calc, fpkm, fractionLength, initiationScore, insideOutsideORF, isInFrame, isOverlapping, kozakSequenceScore, orfScore, rankOrder, ribosomeReleaseScore, ribosomeStallingScore, startRegionCoverage, startRegion, subsetCoverage, translationalEff

Examples

```
 # a small example without cage-seq data:
 # we will find ORFs in the 5' utrs
 # and then calculate features on them
 ## Not run:
 if (requireNamespace("BSgenome.Hsapiens.UCSC.hg19")) {
  library(GenomicFeatures)
  # Get the gtf txdb file
  txdbFile <- system.file("extdata", "hg19_knownGene_sample.sqlite",
  package = "GenomicFeatures")
  txdb <- loadDb(txdbFile)

  # Extract sequences of fiveUTRs.
  fiveUTRs <- fiveUTRsByTranscript(txdb, use.names = TRUE)[1:10]
  faFile <- BSgenome.Hsapiens.UCSC.hg19::Hsapiens
  # need to suppress warning because of bug in GenomicFeatures, will
  # be fixed soon.
  tx_seqs <- suppressWarnings(extractTranscriptSeqs(faFile, fiveUTRs))

  # Find all ORFs on those transcripts and get their genomic coordinates
  fiveUTR_ORFs <- findMapORFs(fiveUTRs, tx_seqs)
  unlistedORFs <- unlistGrl(fiveUTR_ORFs)
  # group GRanges by ORFs instead of Transcripts
  fiveUTR_ORFs <- groupGRangesBy(unlistedORFs, unlistedORFs$names)

  # make some toy ribo seq and rna seq data
  starts <- unlistGrl(ORFik:::firstExonPerGroup(fiveUTR_ORFs))
  RFP <- promoters(starts, upstream = 0, downstream = 1)
  score(RFP) <- rep(29, length(RFP)) # the original read widths

  # set RNA seq to duplicate transcripts
  RNA <- unlistGrl(exonsBy(txdb, by = "tx", use.names = TRUE))

  computeFeaturesCage(grl = fiveUTR_ORFs, orfFeatures =  TRUE, RFP = RFP,
   RNA = RNA, Gtf = txdb, faFile = faFile)

}
# See vignettes for more examples

## End(Not run)
```

`convertToOneBasedRanges` *Convert a GRanges Object to 1 width reads*

Description

There are 4 ways of doing this 1. Take 5' ends, reduce away rest (5prime) 2. Take 3' ends, reduce away rest (3prime) 3. Tile to 1-mers and include all (tileAll) 4. Take middle point per GRanges (middle)

Usage

```
convertToOneBasedRanges(gr, method = "5prime", addScoreColumn = FALSE,
  addSizeColumn = FALSE, after.softclips = TRUE)
```

Arguments

gr	GRanges, GAlignment Object to reduce
method	the method to reduce, see info. (5prime defualt)
addScoreColumn	logical (FALSE), if TRUE, add a score column that sums up the hits per unique range This will make each read unique, so that each read is 1 time, and score column gives the number of hits. A useful compression.
addSizeColumn	logical (FALSE), if TRUE, add a size column that for each read, that gives original width of read. Useful if you need original read lengths. This takes care of soft clips etc.
after.softclips	logical (TRUE), TRUE: include softclips in size

Details

Many other ways to do this have their own functions, like startSites and stopSites etc. To retain information on original width, set addSizeColumn to TRUE. To compress data, 1 GRanges object per unique read, set addScoreColumn to TRUE. This will give you a score column with how many duplicated reads there were in the specified region.

NOTE: Does not support paired end reads for the moment!

Value

Converted GRanges object

See Also

Other utils: bedToGR, findFa, fread.bed, optimizeReads, readBam, readWig

---

coverageGroupings *Get grouping for a coverage table in ORFik*

Description

Either of two groupings: GF: Gene, fraction FGF: Fraction, position, feature It finds which of these exists, and auto groups

Usage

```
coverageGroupings(logicals, grouping = "GF")
```

Arguments

logicals	size 2 logical vector, the is.null checks for each column,
grouping	which grouping to perform

Details

Normally not used directly

Value

a quote of the grouping to pass to data.table

`coverageHeatMap` *Create a heatmap of coverage*

Description

Rows: Position in region Columns: Read length Index intensity: (color) coverage scoring per index.

Usage

```
coverageHeatMap(coverage, output = NULL, scoring = "zscore",
  legendPos = "right", addFracPlot = FALSE)
```

Arguments

`coverage`	a data.table, e.g. output of scaledWindowCoverage
`output`	character string (NULL), if set, saves the plot as pdf or png to path given. If no format is given, is save as pdf.
`scoring`	character vector (zscore), either of zScore, transcriptNormalized, sum, mean, median, NULL. Set NULL if already scored.
`legendPos`	a character, Default "right". Where should the fill legend be ? ("top", "bottom", "right", "left")
`addFracPlot`	Add plot on top of heatmap with fractions per positions

Details

Coverage rows in heat map is fraction, usually fractions is divided into unique read lengths (standard Illumina is 76 unique widths, with some minimum cutoff like 15.) Coverage column in heat map is score, default zscore of counts. These are the relative positions you are plotting to. Like +/- relative to TIS or TSS.

Colors: Remember if you want to change anything like colors, just return the ggplot object, and reassign like: obj + scale_color_brewer() etc. Standard colors are: 0 reads in whole readlength: gray few reads in position: white medium reads in position: yellow many reads in position: dark blue

Value

a ggplot object of the coverage plot, NULL if output is set, then the plot will only be saved to location.

See Also

Other coveragePlot: `pSitePlot`, `savePlot`, `windowCoveragePlot`

Examples

```
# An ORF
grl <- GRangesList(tx1 = GRanges("1", IRanges(1, 6), "+"))
# Ribo-seq reads
range <- IRanges(c(rep(1, 3), 2, 3, rep(4, 2), 5, 6), width = 1 )
reads <- GRanges("1", range, "+")
reads$size <- c(rep(28, 5), rep(29, 4)) # read size
coverage <- ORFik:::windowPerReadLength(grl, reads = reads, upstream = 0,
                                        downstream = 5)

coverageHeatMap(coverage)

# See vignette for more examples
```

`coveragePerTiling` *Get coverage per group*

Description

It tiles each GRangesList group, and finds hits per position

Usage

```
coveragePerTiling(grl, reads, is.sorted = FALSE, keep.names = TRUE,
  as.data.table = FALSE, withFrames = FALSE)
```

Arguments

`grl`	a `GRangesList` of 5' utrs or transcripts.
`reads`	a GAlignment or GRanges object of RiboSeq, RnaSeq etc.
`is.sorted`	logical (F), is grl sorted.
`keep.names`	logical (T), keep names or not.
`as.data.table`	a logical (FALSE), return as data.table with 2 columns, position and count.
`withFrames`	a logical (FALSE), only available if as.data.table is TRUE, return the ORF frame, 1,2,3, where position 1 is 1, 2 is 2 and 4 is 1 etc.

Details

This is a safer speedup of coverageByTranscript from GenomicFeatures. It also gives the possibility to return as data.table, for faster computations.

Value

a RleList, one integer-Rle per group with # of hits per position. Or data.table if as.data.table is TRUE.

See Also

Other ExtendGenomicRanges: `asTX`, `overlapsToCoverage`, `reduceKeepAttr`, `tile1`, `txSeqsFromFa`, `windowPerGroup`

Examples

```
ORF <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(1, 10, 20),
                                end = c(5, 15, 25)),
               strand = "+")
grl <- GRangesList(tx1_1 = ORF)
RFP <- GRanges("1", IRanges(25, 25), "+")
coveragePerTiling(grl, RFP, is.sorted = TRUE)
# now as data.table with frames
coveragePerTiling(grl, RFP, is.sorted = TRUE, as.data.table = TRUE,
                  withFrames = TRUE)
```

`coverageScorings` *Add a coverage scoring scheme*

Description

Different scorings and groupings of a coverage representation.

Usage

```
coverageScorings(coverage, scoring = "zscore")
```

Arguments

`coverage`	a data.table containing at least columns (count, position), it is possible to have additionals: (genes, fraction, feature)
`scoring`	a character, one of (zscore, transcriptNormalized, mean, median, sum, log2sum, log10sum, sumLength, meanPos and frameSum, periodic, NULL). More info in docs.

Details

Usually output of metaWindow or scaledWindowCoverage is input in this function.

Content of coverage data.table: It must contain the count and position columns.

genes column: If you have multiple windows, the genes column must define which gene/transcript grouping the different counts belong to. If there is only a meta window or only 1 gene/transcript, then this column is not needed.

fraction column: If you have coverage of i.e RNA-seq and Ribo-seq, or TCP -seq of large and small subunite, divide into fractions. Like factor(RNA, RFP)

feature column: If gene group is subdivided into parts, like gene is transcripts, and feature column can be c(leader, cds, trailer) etc.

Given a data.table coverage of counts, add a scoring scheme. per: the grouping given, if genes is defined, group by per gene in scoring. Scorings: 1. zscore (count-windowMean)/windowSD per) 2. transcriptNormalized (sum(count / sum of counts per)) 3. mean (mean(count per)) 4. median (median(count per)) 5. sum (count per) 6. log2sum (count per) 7. log10sum (count per) 8. sumLength (count per) / number of windows 9. meanPos (mean per position per gene) used in scaledWindowPositions 10. sumPos (sum per position per gene) used in scaledWindowPositions 11. frameSum (sum per frame per gene) used in ORFScore 12. fracPos (fraction of counts per position per gene) 13. periodic (Fourier transform periodicity of meta coverage per fraction) 14. NULL (return input directly)

Value

a data.table with new scores (size dependent on score used)

See Also

Other coverage: metaWindow, scaledWindowPositions, windowPerReadLength

Examples

```
dt <- data.table::data.table(count = c(4, 1, 1, 4, 2, 3),
                             position = c(1, 2, 3, 4, 5, 6))
coverageScorings(dt, scoring = "zscore")

# with grouping gene
dt$genes <- c(rep("tx1", 3), rep("tx2", 3))
coverageScorings(dt, scoring = "zscore")
```

---

create.experiment *Create a template for new ORFik experiment*

Description

By using files in a folder. It will try to make an experiment table with information per sample. You will have to fill in the details that were not autodetected. Easiest to do in csv editor like libre Office or excel. Remember that each row (sample) must have a unique combination of values.

Usage

```
create.experiment(dir, exper, saveDir = NULL, types = c("bam", "bed",
  "wig"), txdb = "", fa = "", viewTemplate = TRUE)
```

Arguments

dir	Which directory to create experiment from
exper	Short name of experiment, max 5 characters long
saveDir	Directory to save experiment csv file (NULL)
types	Default (bam, bed, wig), which types of libraries to allow
txdb	A path to gff/gtf file used for libraries
fa	A path to fasta genome/sequences used for libraries
viewTemplate	run View() on template when finished, default (TRUE)

Value

a data.frame, NOTE: this is not a ORFik experiment, only a template for it!

Examples

```
template <- create.experiment(dir = system.file("extdata", "", package = "ORFik"),
                              exper = "ORFik", txdb = system.file("extdata",
                                    "annotations.gtf",
                                    package = "ORFik"),
                              viewTemplate = FALSE)
template$X5[6] <- "heart" # <- fix non unique row
# read experiment
df <- read.experiment(template)
```

`defineIsoform` *Overlaps GRanges object with provided annotations.*

Description

Overlaps GRanges object with provided annotations.

Usage

```
defineIsoform(rel_orf, tran, isoform_names = c("perfect_match",
  "elong_START_match", "trunc_START_match", "elong_STOP_match",
  "trunc_STOP_match", "overlap_inside", "overlap_both", "overlap_upstream",
  "overlap_downstream", "upstream", "downstram", "none"))
```

Arguments

`rel_orf`	- GRanges object of your ORF.
`tran`	- GRanges object of annotation (transcript or cds) that overlapped in some way rel_orf.
`isoform_names`	- A vector of strings that will be used instead of these defaults: 'perfect_match' - start and stop matches the tran object strand wise 'elong_START_match' - rel_orf is extension from the STOP side of the tran 'trunc_START_match' - rel_orf is truncation from the STOP side of the tran 'elong_STOP_match' - rel_orf is extension from the START side of the tran 'trunc_STOP_match' - rel_orf is truncation from the START side of the tran 'overlap_inside' - rel_orf is inside tran object 'overlap_both' - rel_orf contains tran object inside 'overlap_upstream' - rel_orf is overlaping upstream part of the tran 'overlap_downstream' - rel_orf is overlaping downstream part of the tran 'upstream' - rel_orf is upstream towards the tran 'downstream' - rel_orf is downstream towards the tran 'none' - when none of the above options is true

Value

A string object of defined isoform towards transcript.

`defineTrailer` *Defines trailers for ORF.*

Description

Creates GRanges object as a trailer for ORFranges representing ORF, maintaining restrictions of transcriptRanges. Assumes that ORFranges is on the transcriptRanges, strands and seqlevels are in agreement. When lengthOFtrailer is smaller than space left on the transcript than all available space is returned as trailer.

Usage

```
defineTrailer(ORFranges, transcriptRanges, lengthOftrailer = 200)
```

Arguments

`ORFranges`	GRanges object of your Open Reading Frame.
`transcriptRanges`	GRanges object of transtript.
`lengthOftrailer`	Numeric. Default is 10.

Details

It assumes that ORFranges and transcriptRanges are not sorted when on minus strand. Should be like: (200, 600) (50, 100)

Value

A GRanges object of trailer.

See Also

Other ORFHelpers: `longestORFs`, `mapToGRanges`, `orfID`, `startCodons`, `startSites`, `stopCodons`, `stopSites`, `txNames`, `uniqueGroups`, `uniqueOrder`

Examples

```
ORFranges <- GRanges(seqnames = Rle(rep("1", 3)),
                     ranges = IRanges(start = c(1, 10, 20),
                                      end = c(5, 15, 25)),
                     strand = "+")
transcriptRanges <- GRanges(seqnames = Rle(rep("1", 5)),
                            ranges = IRanges(start = c(1, 10, 20, 30, 40),
                                             end = c(5, 15, 25, 35, 45)),
                     strand = "+")
defineTrailer(ORFranges, transcriptRanges)
```

`detectRibosomeShifts` *Detect ribosome shifts*

Description

Utilizes periodicity measurement (fourier transform) and change point analysis to detect ribosomal footprint shifts for each of the ribosomal read lengths. Returns subset of read lengths and their shifts for which top covered transcripts follow periodicity measure. Each shift value assumes 5' anchoring of the reads, so that output offsets values will shift 5' anchored footprints to be on the p-site of the ribosome.

Usage

```
detectRibosomeShifts(footprints, txdb, start = TRUE, stop = FALSE,
  top_tx = 10L, minFiveUTR = 30L, minCDS = 150L, minThreeUTR = 30L,
  firstN = 150L, tx = NULL)
```

Arguments

`footprints`	(GAlignments) object of RiboSeq reads - footprints, can also be path to the file.
`txdb`	a TxDb file, an ORFik experiment or a path to one of: (.gtf ,.gff, .gff2, .gff2, .db or .sqlite)
`start`	(logical) Whether to include predictions based on the start codons. Default TRUE.
`stop`	(logical) Whether to include predictions based on the stop codons. Default FASLE. Only use if there exists 3' UTRs for the annotation.
`top_tx`	(integer) Specify which transcripts to use for estimation of the shifts. By default we take top 10 top covered transcripts as they represent less noisy dataset. This is only applicable when there are more than 1000 transcripts.
`minFiveUTR`	(integer) minimum bp for 5' UTR during filtering for the transcripts. Set to NULL if no 5' UTRs exists for annotation.
`minCDS`	(integer) minimum bp for CDS during filtering for the transcripts
`minThreeUTR`	(integer) minimum bp for 3' UTR during filtering for the transcripts. Set to NULL if no 3' UTRs exists for annotation.
`firstN`	(integer) Represents how many bases of the transcripts downstream of start codons to use for initial estimation of the periodicity.
`tx`	a GRangesList, if you do not have 5' UTRs in annotation, send your own version. Example: extendLeaders(tx, 30) Where 30 bases will be new "leaders". Since each original transcript was either only CDS or non-coding (filtered out).

Details

Check out vignette for the examples of plotting RiboSeq metaplots over start and stop codons, so that you can verify visually whether this function detects correct shifts.

NOTE: It will remove softclips from valid width, the CIGAR 3S30M is qwidth 33, but will remove 3S so final read width is 30 in ORFik.

Value

a data.table with lengths of footprints and their predicted coresponding offsets

See Also

Other pshifting: changePointAnalysis, shiftFootprints

Examples

```
## Not run:
# Transcriptome annotation ->
gtf_file <- system.file("extdata", "annotations.gtf", package = "ORFik")
# The ribo seq file, usually .bam file ->
riboSeq_file <- system.file("extdata", "ribo-seq.bam", package = "ORFik")
footprints <- GenomicAlignments::readGAlignments(
  riboSeq_file, param = ScanBamParam(flag = scanBamFlag(
    isDuplicate = FALSE, isSecondaryAlignment = FALSE)))

detectRibosomeShifts(footprints, gtf_file, stop = TRUE)

# Without 5' Annotation
library(GenomicFeatures)

txdb <- loadTxdb(gtf_file)
tx <- exonsBy(txdb, by = "tx", use.names = TRUE)
tx <- extendLeaders(tx, 30)
# Now run function, without 5' and 3' UTRs
detectRibosomeShifts(footprints, txdb, start = TRUE, minFiveUTR = NULL,
                     minCDS = 150L, minThreeUTR = NULL, firstN = 150L,
                     tx = tx)


## End(Not run)
```

---

disengagementScore *Disengagement score (DS)*

Description

Disengagement score is defined as

```
(RPFs over ORF)/(RPFs downstream to transcript end)
```

A pseudo-count of one is added to both the ORF and downstream sums.

Usage

```
disengagementScore(grl, RFP, GtfOrTx, RFP.sorted = FALSE)
```

Arguments

grl	a GRangesList object with usually either leaders, cds', 3' utrs or ORFs.
RFP	RiboSeq reads as GAlignment, GRanges or GRangesList object
GtfOrTx	If it is TxDb object transcripts will be extracted using exonsBy(Gtf,by = "tx",use.names = TRUE). Else it must be GRangesList
RFP.sorted	logical (F), an optimizer, have you ran this line: RFP <-sort(RFP[countOverlaps(RFP,tx,type = "within") > 0]) Normally not touched, for internal optimization purposes.

Value

a named vector of numeric values of scores

References

doi: 10.1242/dev.098344

See Also

Other features: computeFeaturesCage, computeFeatures, distToCds, distToTSS, entropy, floss, fpkm_calc, fpkm, fractionLength, initiationScore, insideOutsideORF, isInFrame, isOverlapping, kozakSequenceScore, orfScore, rankOrder, ribosomeReleaseScore, ribosomeStallingScore, startRegionCoverage, startRegion, subsetCoverage, translationalEff

Examples

```
ORF <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(1, 10, 20), end = c(5, 15, 25)),
               strand = "+")
grl <- GRangesList(tx1_1 = ORF)
tx <- GRangesList(tx1 = GRanges("1", IRanges(1, 50), "+"))
RFP <- GRanges("1", IRanges(c(1,10,20,30,40), width = 3), "+")
disengagementScore(grl, RFP, tx)
```

---

distToCds *Get distances between ORF ends and starts of their transcripts cds.*

Description

Will calculate distance between each ORF end and begining of the corresponding cds (main ORF). Matching is done by transcript names. This is applicable practically to the upstream (fiveUTRs) ORFs only. The cds start site, will be presumed to be on + 1 of end of fiveUTRs.

Usage

```
distToCds(ORFs, fiveUTRs, cds = NULL)
```

Arguments

ORFs	orfs as GRangesList, names of orfs must be transcript names
fiveUTRs	fiveUTRs as GRangesList, remember to use CAGE version of 5' if you did CAGE reassignment!
cds	cds' as GRangesList, only add if you have ORFs going into CDS.

Value

an integer vector, +1 means one base upstream of cds, -1 means 2nd base in cds, 0 means orf stops at cds start.

References

doi: 10.1074/jbc.R116.733899

See Also

Other features: computeFeaturesCage, computeFeatures, disengagementScore, distToTSS, entropy, floss, fpkm_calc, fpkm, fractionLength, initiationScore, insideOutsideORF, isInFrame, isOverlapping, kozakSequenceScore, orfScore, rankOrder, ribosomeReleaseScore, ribosomeStallingScore, startRegionCoverage, startRegion, subsetCoverage, translationalEff

Examples

```
grl <- GRangesList(tx1_1 = GRanges("1", IRanges(1, 10), "+"))
fiveUTRs <- GRangesList(tx1 = GRanges("1", IRanges(1, 20), "+"))
distToCds(grl, fiveUTRs)
```

---

distToTSS *Get distances between ORF Start and TSS of its transcript*

Description

Matching is done by transcript names. This is applicable practically to any region in Transcript If ORF is not within specified search space in tx, this function will crash.

Usage

```
distToTSS(ORFs, tx)
```

Arguments

ORFs	orfs as GRangesList, names of orfs must be txname_[rank]
tx	transcripts as GRangesList.

Value

an integer vector, 1 means on TSS, 2 means second base of Tx.

References

doi: 10.1074/jbc.R116.733899

See Also

Other features: computeFeaturesCage, computeFeatures, disengagementScore, distToCds, entropy, floss, fpkm_calc, fpkm, fractionLength, initiationScore, insideOutsideORF, isInFrame, isOverlapping, kozakSequenceScore, orfScore, rankOrder, ribosomeReleaseScore, ribosomeStallingScore, startRegionCoverage, startRegion, subsetCoverage, translationalEff

Examples

```
grl <- GRangesList(tx1_1 = GRanges("1", IRanges(5, 10), "+"))
tx <- GRangesList(tx1 = GRanges("1", IRanges(2, 20), "+"))
distToTSS(grl, tx)
```

---

downstreamFromPerGroup

Get rest of objects downstream (inclusive)

Description

Per group get the part downstream of position. downstreamFromPerGroup(tx, startSites(threeUTRs, asGR = TRUE)) will return the 3' utrs per transcript as GRangesList, usually used for interesting parts of the transcripts.

Usage

```
downstreamFromPerGroup(tx, downstreamFrom)
```

Arguments

tx	a GRangesList, usually of Transcripts to be changed
downstreamFrom	a vector of integers, for each group in tx, where is the new start point of first valid exon.

Details

If you don't want to include the points given in the region, use downstreamOfPerGroup

Value

a GRangesList of downstream part

See Also

Other GRanges: assignFirstExonsStartSite, assignLastExonsStopSite, downstreamOfPerGroup, upstreamFromPerGroup, upstreamOfPerGroup

`downstreamN` *Restrict GRangesList*

Description

Will restrict GRangesList to 'N' bp downstream from the first base.

Usage

```
downstreamN(grl, firstN = 150L)
```

Arguments

`grl`	(GRangesList)
`firstN`	(integer) Allow only this many bp downstream, maximum.

Value

a GRangesList of reads restricted to firstN and tiled by 1

`downstreamOfPerGroup` *Get rest of objects downstream (exclusive)*

Description

Per group get the part downstream of position. downstreamOfPerGroup(tx, stopSites(cds, asGR = TRUE)) will return the 3' utrs per transcript as GRangesList, usually used for interesting parts of the transcripts.

Usage

```
downstreamOfPerGroup(tx, downstreamOf)
```

Arguments

`tx`	a `GRangesList`, usually of Transcripts to be changed
`downstreamOf`	a vector of integers, for each group in tx, where is the new start point of first valid exon.

Details

If you want to include the points given in the region, use downstreamFromPerGroup

Value

a GRangesList of downstream part

See Also

Other GRanges: `assignFirstExonsStartSite`, `assignLastExonsStopSite`, `downstreamFromPerGroup`, `upstreamFromPerGroup`, `upstreamOfPerGroup`

entropy *Calucalte entropy value of overlapping input reads per GRanges.*

Description

Calculates entropy of the 'reads' coverage over each 'grl' group. The entropy value per group is a real number in the interval (0:1), where 0 indicates no variance in reads over group. For example c(0,0,0,0) has 0 entropy, since no reads overlap.

Usage

```
entropy(grl, reads)
```

Arguments

grl	a GRangesList that the reads will be overlapped with
reads	a GAlignment, GRanges or GRangesList object, usually of RiboSeq, RnaSeq, CageSeq, etc.

Value

A numeric vector containing one entropy value per element in 'grl'

See Also

Other features: computeFeaturesCage, computeFeatures, disengagementScore, distToCds, distToTSS, floss, fpkm_calc, fpkm, fractionLength, initiationScore, insideOutsideORF, isInFrame, isOverlapping, kozakSequenceScore, orfScore, rankOrder, ribosomeReleaseScore, ribosomeStallingScore, startRegionCoverage, startRegion, subsetCoverage, translationalEff

Examples

```
# a toy example with ribo-seq p-shifted reads
ORF <- GRanges("1", ranges = IRanges(start = c(1, 12, 22),
                                     end = c(10, 20, 32)),
               strand = "+",
               names = rep("tx1_1", 3))
names(ORF) <- rep("tx1", 3)
grl <- GRangesList(tx1_1 = ORF)
reads <- GRanges("1", IRanges(c(25, 35), c(25, 35)), "+")
# grl must have same names as cds + _1 etc, so that they can be matched.
entropy(grl, reads)
# or on cds
cdsORF <- GRanges("1", IRanges(35, 44), "+", names = "tx1")
names(cdsORF) <- "tx1"
cds <-  GRangesList(tx1 = cdsORF)
entropy(cds, reads)
```

experiment-class *experiment class definition*

Description

An object to massivly simplify your coding, it is similar to systempipeR's 'target' table. By containing filepaths and info for each library in some experiment.

Details

Simplest way to make is to call create.experiment on some folder with libraries and see what you get. Some of the fields might be needed to fill in manually. The important thing is that each row must be unique (excluding filepath), that means if it has replicates then that must be said explicit. And all filepaths must be unique and have files with size > 0. Syntax: libtype (library type): rna-seq, ribo-seq, CAGE etc. rep (replicate): 1,2,3 etc condition: WT (wild-type), control, target, mzdicer, starved etc. fraction: 18, 19 (fractinations), or other ways to split library. filepath: Full filepath to file

extendLeaders *Extend the leaders transcription start sites.*

Description

Will extend the leaders or transcripts upstream by extension. Remember the extension is general not relative, that means splicing will not be taken into account. Requires the `grl` to be sorted beforehand, use `sortPerGroup` to get sorted grl.

Usage

```
extendLeaders(grl, extension = 1000L, cds = NULL)
```

Arguments

`grl`	usually a `GRangesList` of 5' utrs or transcripts. Can be used for any extension of groups.
`extension`	an integer, how much to extend the leaders. Or a GRangesList where start / stops by strand are the positions to use as new starts.
`cds`	If you want to extend 5' leaders downstream, to catch upstream ORFs going into cds, include it. It will add first cds exon to grl matched by names. Do not add for transcripts, as they are already included.

Value

an extended GRangeslist

Examples

```
library(GenomicFeatures)
samplefile <- system.file("extdata", "hg19_knownGene_sample.sqlite",
                          package = "GenomicFeatures")
txdb <- loadDb(samplefile)
fiveUTRs <- fiveUTRsByTranscript(txdb) # <- extract only 5' leaders
tx <- exonsBy(txdb, by = "tx", use.names = TRUE)
cds <- cdsBy(txdb,"tx",use.names = TRUE)
## now try(extend upstream 1000, downstream 1st cds exons):
extendLeaders(fiveUTRs, extension = 1000, cds)

## when extending transcripts, don't include cds' of course,
## since they are already there
extendLeaders(tx, extension = 1000)
```

---

extendsTSSexons *Extend first exon of each transcript with length specified*

Description

Extend first exon of each transcript with length specified

Usage

```
extendsTSSexons(fiveUTRs, extension = 1000)
```

Arguments

fiveUTRs	The 5' leader sequences as GRangesList
extension	The number of basses to extend transcripts upstream

Value

GRangesList object of fiveUTRs

---

extendTrailers *Extend the Trailers transcription stop sites*

Description

Will extend the trailers or transcripts downstream by extension. Remember the extension is general not relative, that means splicing will not be taken into account. Requires the grl to be sorted beforehand, use sortPerGroup to get sorted grl.

Usage

```
extendTrailers(grl, extension = 1000L)
```

Arguments

grl	usually a GRangesList of 3' utrs or transcripts. Can be used for any extension of groups.
extension	an integer, how much to extend the leaders. Or a GRangesList where start / stops by strand are the positions to use as new starts.

Value

an extended GRangeslist

Examples

```
library(GenomicFeatures)
samplefile <- system.file("extdata", "hg19_knownGene_sample.sqlite",
                          package = "GenomicFeatures")
txdb <- loadDb(samplefile)
threeUTRs <- threeUTRsByTranscript(txdb) # <- extract only 5' leaders
tx <- exonsBy(txdb, by = "tx", use.names = TRUE)
## now try(extend downstream 1000):
extendTrailers(threeUTRs, extension = 1000)
## Or on transcripts
extendTrailers(tx, extension = 1000)
```

---

filterCage *Filter peak of cage-data by value*

Description

Filter peak of cage-data by value

Usage

```
filterCage(cage, filterValue = 1, fiveUTRs = NULL, preCleanup = TRUE)
```

Arguments

cage	Either a filePath for the CageSeq file as .bed .bam or .wig, with possible compressions (".gzip", ".gz", ".bgz"), or already loaded CageSeq peak data as GRanges or GAlignment. NOTE: If it is a .bam file, it will add a score column by running: convertToOneBasedRanges(cage, method = "5prime", addScoreColumn = TRUE)
filterValue	The minimum number of reads on cage position, for it to be counted as possible new tss. (represented in score column in CageSeq data) If you already filtered, set it to 0.
fiveUTRs	a GRangesList (NULL), if added will filter out cage reads by these following rules: all reads in region (-5:-1, 1:5) for each tss will be removed, removes noise.
preCleanup	logical (TRUE), if TRUE, remove all reads in region (-5:-1, 1:5) of all original tss in leaders. This is to keep original TSS if it is only +/- 5 bases from the original.

Value

the filtered Granges object

`filterTranscripts` *Filter transcripts by lengths*

Description

Filter transcripts to those who have leaders, CDS, trailers of some lengths, you can also pick the longest per gene.

Usage

```
filterTranscripts(txdb, minFiveUTR = 30L, minCDS = 150L,
  minThreeUTR = 30L, longestPerGene = TRUE, stopOnEmpty = TRUE)
```

Arguments

`txdb`	a TxDb file, an ORFik experiment or a path to one of: (.gtf ,.gff, .gff2, .gff2, .db or .sqlite)
`minFiveUTR`	(integer) minimum bp for 5' UTR during filtering for the transcripts. Set to NULL if no 5' UTRs exists for annotation.
`minCDS`	(integer) minimum bp for CDS during filtering for the transcripts
`minThreeUTR`	(integer) minimum bp for 3' UTR during filtering for the transcripts. Set to NULL if no 3' UTRs exists for annotation.
`longestPerGene`	logical (TRUE), return only longest valid transcript per gene.
`stopOnEmpty`	logical TRUE, stop if no valid transcripts are found ?

Details

If a transcript does not have a trailer, then the length is 0, so they will be filtered out if you set minThreeUTR to 1. So only transcripts with leaders, cds and trailers will be returned. You can set the integer to 0, that will return all within that group.

If your annotation does not have leaders or trailers, set them to NULL.

Value

a character vector of valid transcript names

Examples

```
gtf_file <- system.file("extdata", "annotations.gtf", package = "ORFik")
txdb <- GenomicFeatures::makeTxDbFromGFF(gtf_file)
txNames <- filterTranscripts(txdb, minFiveUTR = 1, minCDS = 30,
                             minThreeUTR = 1)
loadRegion(txdb, "mrna")[txNames]
loadRegion(txdb, "5utr")[txNames]
```

filterUORFs *Remove uORFs that are false CDS hits*

Description

This is a strong filtering, so that even if the cds is on another transcript , the uORF is filtered out, this is because there is no way of knowing by current ribo-seq, rna-seq experiments.

Usage

```
filterUORFs(uorfs, cds)
```

Arguments

uorfs	(GRangesList), the uORFs to filter
cds	(GRangesList), the coding sequences (main ORFs on transcripts), to filter against.

Value

(GRangesList) of filtered uORFs

See Also

Other uorfs: addCdsOnLeaderEnds, removeORFsWithSameStartAsCDS, removeORFsWithSameStopAsCDS, removeORFsWithStartInsideCDS, removeORFsWithinCDS, uORFSearchSpace

fimport *Load any type of sequencing reads*

Description

Wraps around rtracklayer::import and tries to speed up loading with the use of data.table. Supports gzip, gz, bgz compression formats. Also safer chromosome naming with the argument chrStyle

Usage

```
fimport(path, chrStyle = NULL)
```

Arguments

path	a character path to file or a GRanges/Galignment object etc. Any Ranged object.
chrStyle	a GRanges object, or a character style (Default: NULL) to get seqlevelsStyle from. Is chromosome 1 called chr1 or 1, is mitocondrial chromosome called MT or chrM etc. Will use 1st seqlevel- style if more are present. Like: c("NCBI", "UCSC") -> pick "NCBI"

Details

NOTE: For wig you can send in 2 files, so that it automaticly merges forward and reverse stranded objects. You can also just send 1 wig file, it will then have "*" as strand.

Value

a GAlignment/GRanges object depending on input.

`findFa` *Convenience wrapper for Rsamtools FaFile*

Description

Convenience wrapper for Rsamtools FaFile

Usage

```
findFa(faFile)
```

Arguments

`faFile`	FaFile, BSgenome or fasta/index file path used to find the transcripts

Value

a FaFile or BSgenome

See Also

Other utils: `bedToGR`, `convertToOneBasedRanges`, `fread.bed`, `optimizeReads`, `readBam`, `readWig`

`findFromPath` *Find all candidate library types filenames*

Description

Find all candidate library types filenames

Usage

```
findFromPath(filepaths, candidates = c("RNA", "rna-seq", "Rna-seq",
  "RNA-seq", "RFP", "RPF", "ribo-seq", "Ribo-seq", "mrna", "mrna-seq",
  "mRNA-seq", "CAGE", "cage", "LSU", "SSU", "ATAC", "tRNA", "SHAPE"))
```

Arguments

`filepaths`	path to all files
`candidates`	Possible names to search for.

Value

a candidate library types (character vector)

findLibrariesInFolder *Get all library files in folder*

Description

Get all library files in folder

Usage

```
findLibrariesInFolder(dir, types)
```

Arguments

dir	The directory to find bam, bed, wig files.
types	All accepted types of bam, bed, wig files..

Value

(character vector) All files found from types parameter.

findMapORFs *Find ORFs and immediately map them to their genomic positions.*

Description

Finds ORFs on the sequences of interest, but returns relative positions to the positions of 'grl' argument. For example, 'grl' can be exons of known transcripts (with genomic coordinates), and 'seq' sequences of those transcripts, in that case, this function will return genomic coordinates of ORFs found on transcript sequences.

Usage

```
findMapORFs(grl, seqs, startCodon = startDefinition(1),
  stopCodon = stopDefinition(1), longestORF = TRUE,
  minimumLength = 0, groupByTx = TRUE)
```

Arguments

grl	(GRangesList) of sequences to search for ORFs, probably in genomic coordinates
seqs	(DNAStringSet or character vector) - DNA/RNA sequences to search for Open Reading Frames. Can be both uppercase or lowercase. Easiest call to get seqs if you want only regions from a fasta/fasta index pair is: seqs = ORFik:::txSeqsFromFa(grl, faFile), where grl is a GRanges/List of search regions and faFile is a FaFile.
startCodon	(character vector) Possible START codons to search for. Check startDefinition for helper function.
stopCodon	(character vector) Possible STOP codons to search for. Check stopDefinition for helper function.

longestORF	(logical) Default TRUE. Keep only the longest ORF per unique (seqname, strand, stopcodon) combination, you can also use function longestORFs after creation of ORFs for same result.
minimumLength	(integer) Default is 0. Which is START + STOP = 6 bp. Minimum length of ORF, without counting 3bp for START and STOP codons. For example minimumLength = 8 will result in size of ORFs to be at least START + 8*3 (bp) + STOP = 30 bases. Use this param to restrict search.
groupByTx	logical (T), should output GRangesList be grouped by orfs per transcript (T) or by exons per ORF (F)?

Details

This function assumes that 'seq' is in widths relative to 'grl', and that their orders match. 1st seq is 1st grl object, etc.

See vignette for real life example.

Value

A GRangesList of ORFs.

See Also

Other findORFs: findORFsFasta, findORFs, findUORFs, startDefinition, stopDefinition

Examples

```
# This sequence has ORFs at 1-9 and 4-9
seqs <- c("ATGATGTAA") # the dna sequence
findORFs(seqs)
# lets assume that this sequence comes from two exons as follows
gr <- GRanges(seqnames = rep("1", 2), # chromosome 1
              ranges = IRanges(start = c(21, 10), end = c(23, 15)),
              strand = rep("-", 2), names = rep("tx1", 2))
grl <- GRangesList(tx1 = gr)
findMapORFs(grl, seqs) # ORFs are properly mapped to its genomic coordinates

grl <- c(grl, grl)
names(grl) <- c("tx1", "tx2")
findMapORFs(grl, c(seqs, seqs))
```

---

findMaxPeaks *Find max peak for each transcript, returns as data.table, without names, but with index*

Description

Find max peak for each transcript, returns as data.table, without names, but with index

Usage

```
findMaxPeaks(cageOverlaps, filteredCage)
```

Arguments

cageOverlaps	The cageOverlaps between cage and extended 5' leaders
filteredCage	The filtered raw cage-data used to reassign 5' leaders

Value

a data.table of max peaks

---

findNewTSS *Finds max peaks per trancsript from reads in the cagefile*

Description

Finds max peaks per trancsript from reads in the cagefile

Usage

```
findNewTSS(fiveUTRs, cageData, extension, restrictUpstreamToTx)
```

Arguments

fiveUTRs	The 5' leader sequences as GRangesList
cageData	The CAGE as GRanges object
extension	The number of basses upstream to add on transcripts
restrictUpstreamToTx	a logical (FALSE), if you want to restrict leaders to not extend closer than 5 bases from closest upstream leader, set this to TRUE.

Value

a Hits object

---

findORFs *Find Open Reading Frames.*

Description

Find all Open Reading Frames (ORFs) on the input sequences in ONLY 5'- 3' direction (+), but within all three possible reading frames. For each sequence of the input vector IRanges with START and STOP positions (inclusive) will be returned as IRangesList. Returned coordinates are relative to the input sequences.

Usage

```
findORFs(seqs, startCodon = startDefinition(1),
  stopCodon = stopDefinition(1), longestORF = TRUE,
  minimumLength = 0)
```

Arguments

seqs	(DNAStringSet or character vector) - DNA/RNA sequences to search for Open Reading Frames. Can be both uppercase or lowercase. Easiest call to get seqs if you want only regions from a fasta/fasta index pair is: seqs = ORFik:::txSeqsFromFa(grl, faFile), where grl is a GRanges/List of search regions and faFile is a FaFile.
startCodon	(character vector) Possible START codons to search for. Check startDefinition for helper function.
stopCodon	(character vector) Possible STOP codons to search for. Check stopDefinition for helper function.
longestORF	(logical) Default TRUE. Keep only the longest ORF per unique (seqname, strand, stopcodon) combination, you can also use function longestORFs after creation of ORFs for same result.
minimumLength	(integer) Default is 0. Which is START + STOP = 6 bp. Minimum length of ORF, without counting 3bp for START and STOP codons. For example minimumLength = 8 will result in size of ORFs to be at least START + 8*3 (bp) + STOP = 30 bases. Use this param to restrict search.

Details

If you want antisence strand too, do: `#positive strands pos <-findORFs(seqs) #negative strands (DNAStringSet only if character) neg <-findORFs(reverseComplement(DNAStringSet(seqs))) relist(c(GRanges(pos,strand = "+"),GRanges(neg,strand = "-")),skeleton = merge(pos,neg))`

Value

(IRangesList) of ORFs locations by START and STOP sites grouped by input seqeunces. In a list of sequences, only the indices of the sequences that had ORFs will be returned, e.g. 3 sequences where only 1 and 3 has ORFs, will return size 2 IRangesList with names c("1", "3"). If there are a total of 0 ORFs, an empty IRangesList will be returned.

See Also

Other findORFs: findMapORFs, findORFsFasta, findUORFs, startDefinition, stopDefinition

Examples

```
findORFs("ATGTAA")
findORFs("ATGTTAA") # not in frame anymore

findORFs("ATGATGTAA") # two ORFs
findORFs("ATGATGTAA", longestORF = TRUE) # only longest of two above

findORFs(c("ATGTAA", "ATGATGTAA"))
```

`findORFsFasta` *Finds Open Reading Frames in fasta files.*

Description

Should be used for procaryote genomes or transcript sequences as fasta. Makes no sence for eukaryote whole genomes, since it contains splicing. Searches through each fasta header and reports all ORFs found for BOTH sense (+) and antisense strand (-) in all frames. Name of the header will be used as seqnames of reported ORFs. Each fasta header is treated separately, and name of the sequence will be used as seqname in returned GRanges object. This supports circluar genomes.

Usage

```
findORFsFasta(filePath, startCodon = startDefinition(1),
  stopCodon = stopDefinition(1), longestORF = TRUE,
  minimumLength = 0, is.circular = FALSE)
```

Arguments

`filePath`	(character) Path to the fasta file. Can be both uppercase or lowercase.
`startCodon`	(character vector) Possible START codons to search for. Check `startDefinition` for helper function.
`stopCodon`	(character vector) Possible STOP codons to search for. Check `stopDefinition` for helper function.
`longestORF`	(logical) Default TRUE. Keep only the longest ORF per unique (seqname, strand, stopcodon) combination, you can also use function `longestORFs` after creation of ORFs for same result.
`minimumLength`	(integer) Default is 0. Which is START + STOP = 6 bp. Minimum length of ORF, without counting 3bp for START and STOP codons. For example minimumLength = 8 will result in size of ORFs to be at least START + 8*3 (bp) + STOP = 30 bases. Use this param to restrict search.
`is.circular`	(logical) Whether the genome in filePath is circular. Prokaryotic genomes are usually circular. Be carefull if you want to extract sequences, remember that seqlengths must be set, else it does not know what last base in sequence is before loop ends!

Details

Remember if you have a fasta file of transcripts (transcript coordinates), delete all negative stranded ORFs afterwards by: orfs <- orfs[strandBool(orfs)] # negative strand orfs make no sense then. Seqnames are created from header by format: >name info, so name must be first after "biggern than" and space between name and info.

Value

(GRanges) object of ORFs mapped from fasta file. Positions are relative to the fasta file.

See Also

Other findORFs: `findMapORFs`, `findORFs`, `findUORFs`, `startDefinition`, `stopDefinition`

Examples

```
# location of the example fasta file
example_genome <- system.file("extdata", "genome.fasta", package = "ORFik")
findORFsFasta(example_genome)
```

`findUORFs` *Find upstream ORFs from transcript annotation*

Description

Procedure: 1. Create a new search space starting with the 5' UTRs. 2. Redefine TSS with CAGE if wanted. 3. Add the whole of CDS to search space to allow uORFs going into cds. 4. find ORFs on that search space. 5. Filter out wrongly found uORFs, if CDS is included. The CDS, alternative CDS, uORFs starting within the CDS etc.

Usage

```
findUORFs(fiveUTRs, fa, startCodon = startDefinition(1),
  stopCodon = stopDefinition(1), longestORF = TRUE,
  minimumLength = 0, cds = NULL, cage = NULL, extension = 1000,
  filterValue = 1, restrictUpstreamToTx = FALSE,
  removeUnused = FALSE)
```

Arguments

`fiveUTRs`	(GRangesList) The 5' leaders or full transcript sequences
`fa`	a `FaFile`. With fasta sequences corresponding to fiveUTR annotation. Usually loaded from the genome of an organism with fa = ORFik:::findFa("path/to/fasta/genome")
`startCodon`	(character vector) Possible START codons to search for. Check `startDefinition` for helper function.
`stopCodon`	(character vector) Possible STOP codons to search for. Check `stopDefinition` for helper function.
`longestORF`	(logical) Default TRUE. Keep only the longest ORF per unique (seqname, strand, stopcodon) combination, you can also use function `longestORFs` after creation of ORFs for same result.
`minimumLength`	(integer) Default is 0. Which is START + STOP = 6 bp. Minimum length of ORF, without counting 3bp for START and STOP codons. For example minimumLength = 8 will result in size of ORFs to be at least START + 8*3 (bp) + STOP = 30 bases. Use this param to restrict search.
`cds`	(GRangesList) CDS of relative fiveUTRs, applicable only if you want to extend 5' leaders downstream of CDS's, to allow upstream ORFs that can overlap into CDS's.
`cage`	Either a filePath for the CageSeq file as .bed .bam or .wig, with possible compressions (".gzip", ".gz", ".bgz"), or already loaded CageSeq peak data as GRanges or GAlignment. NOTE: If it is a .bam file, it will add a score column by running: convertToOneBasedRanges(cage, method = "5prime", addScoreColumn = TRUE)

extension	The maximum number of basses upstream of the TSS to search for CageSeq peak.
filterValue	The minimum number of reads on cage position, for it to be counted as possible new tss. (represented in score column in CageSeq data) If you already filtered, set it to 0.
restrictUpstreamToTx	a logical (FALSE). If TRUE: restrict leaders to not extend closer than 5 bases from closest upstream leader, set this to TRUE.
removeUnused	logical (FALSE), if False: (standard is to set them to original annotation), If TRUE: remove leaders that did not have any cage support.

Details

From default a filtering process is done to remove "fake" uORFs, but only if cds is included, since uORFs that stop on the stop codon on the CDS is not a uORF, but an alternative cds by definition.

Value

A GRangesList of uORFs, 1 granges list element per uORF.

See Also

Other findORFs: findMapORFs, findORFsFasta, findORFs, startDefinition, stopDefinition

Examples

```
## Not run:
 # Load annotation
 txdbFile <- system.file("extdata", "hg19_knownGene_sample.sqlite",
                         package = "GenomicFeatures")
 txdb <- loadTxdb(txdbFile)
 fiveUTRs <- loadRegion(txdb, "leaders")
 cds <- loadRegion(txdb, "cds")
 if (requireNamespace("BSgenome.Hsapiens.UCSC.hg19")) {
   # Normally you would not use a BSgenome, but some custom fasta-
   # annotation you  have for your species
   findUORFs(fiveUTRs, BSgenome.Hsapiens.UCSC.hg19::Hsapiens, "ATG",
             cds = cds)
 }

## End(Not run)
```

---

firstEndPerGroup *Get first end per granges group*

Description

grl must be sorted, call ORFik:::sortPerGroup if needed

Usage

```
firstEndPerGroup(grl, keep.names = TRUE)
```

Arguments

grl	a GRangesList
keep.names	a boolean, keep names or not

Value

a Rle(keep.names = T), or integer vector(F)

Examples

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
                   ranges = IRanges(c(7, 14), width = 3),
                   strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
                    ranges = IRanges(c(4, 1), c(9, 3)),
                    strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
firstEndPerGroup(grl)
```

---

firstExonPerGroup *Get first exon per GRangesList group*

Description

grl must be sorted, call ORFik:::sortPerGroup if needed

Usage

```
firstExonPerGroup(grl)
```

Arguments

grl	a GRangesList

Value

a GRangesList of the first exon per group

Examples

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
                   ranges = IRanges(c(7, 14), width = 3),
                   strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
                    ranges = IRanges(c(4, 1), c(9, 3)),
                    strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
firstExonPerGroup(grl)
```

firstStartPerGroup *Get first start per granges group*

Description

grl must be sorted, call ORFik:::sortPerGroup if needed

Usage

```
firstStartPerGroup(grl, keep.names = TRUE)
```

Arguments

grl	a GRangesList
keep.names	a boolean, keep names or not

Value

a Rle(keep.names = TRUE), or integer vector(FALSE)

Examples

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
                   ranges = IRanges(c(7, 14), width = 3),
                   strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
                    ranges = IRanges(c(4, 1), c(9, 3)),
                    strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
firstStartPerGroup(grl)
```

floss *Fragment Length Organization Similarity Score*

Description

This feature is usually calcualted only for RiboSeq reads. For reads of width between 'start' and 'end', sum the fraction of RiboSeq reads (per widths) that overlap ORFs and normalize by CDS.

Usage

```
floss(grl, RFP, cds, start = 26, end = 34)
```

Arguments

grl	a GRangesList object with ORFs
RFP	ribosomal footprints, given as Galignment or GRanges object, must be already shifted and resized to the p-site
cds	a GRangesList of coding sequences, cds has to have names as grl so that they can be matched
start	usually 26, the start of the floss interval
end	usually 34, the end of the floss interval

Details

Pseudo explanation of the function:

```
SUM[start to stop]((grl[start:end][name]/grl) / (cds[start:end][name]/cds))
```

Please read more in the article.

Value

a vector of FLOSS of length same as grl

References

doi: 10.1016/j.celrep.2014.07.045

See Also

Other features: computeFeaturesCage, computeFeatures, disengagementScore, distToCds, distToTSS, entropy, fpkm_calc, fpkm, fractionLength, initiationScore, insideOutsideORF, isInFrame, isOverlapping, kozakSequenceScore, orfScore, rankOrder, ribosomeReleaseScore, ribosomeStallingScore, startRegionCoverage, startRegion, subsetCoverage, translationalEff

Examples

```
ORF <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(1, 12, 22),
               end = c(10, 20, 32)),
               strand = "+")
grl <- GRangesList(tx1_1 = ORF)
# RFP is 1 width position based GRanges
RFP <- GRanges("1", IRanges(c(1, 25, 35, 38), width = 1), "+")
score(RFP) <- c(28, 28, 28, 29) # original width in score col
cds <-  GRangesList(tx1 = GRanges("1", IRanges(35, 44), "+"))
# grl must have same names as cds + _1 etc, so that they can be matched.
floss(grl, RFP, cds)
# or change ribosome start/stop, more strict
floss(grl, RFP, cds, 28, 28)
```

---

fpkm *Create normalizations of overlapping read counts.*

Description

FPKM is short for "Fragments Per Kilobase of transcript per Million fragments in library". When calculating RiboSeq data FPKM over ORFs, use ORFs as ‘grl‘. When calculating RNASeq data FPKM, use full transcripts as ‘grl‘. It is equal to RPKM given that you do not have paired end reads.

Usage

```
fpkm(grl, reads, pseudoCount = 0)
```

Arguments

grl	a GRangesList object can be either transcripts, 5' utrs, cds', 3' utrs or ORFs as a special case (uORFs, potential new cds' etc). If regions are not spliced you can send a GRanges object.
reads	a GAlignment, GRanges or GRangesList object, usually of RiboSeq, RnaSeq, CageSeq, etc.
pseudoCount	an integer, by default is 0, set it to 1 if you want to avoid NA and inf values.

Details

Note also that you must consider if you will use the whole read library or just the reads overlapping 'grl'. To only overlap do: reads <- reads[countOverlaps(reads, grl) > 0]

Value

a numeric vector with the fpkm values

References

doi: 10.1038/nbt.1621

See Also

Other features: computeFeaturesCage, computeFeatures, disengagementScore, distToCds, distToTSS, entropy, floss, fpkm_calc, fractionLength, initiationScore, insideOutsideORF, isInFrame, isOverlapping, kozakSequenceScore, orfScore, rankOrder, ribosomeReleaseScore, ribosomeStallingScore, startRegionCoverage, startRegion, subsetCoverage, translationalEff

Examples

```
ORF <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(1, 10, 20),
               end = c(5, 15, 25)),
               strand = "+")
grl <- GRangesList(tx1_1 = ORF)
RFP <- GRanges("1", IRanges(25, 25),"+")
fpkm(grl, RFP)
```

---

fpkm_calc *Create normalizations of read counts*

Description

A helper for [fpkm()] Normally use function [fpkm()], if you want unusual normalization , you can use this. Short for: Fragments per kilobase of transcript per million fragments Normally used in Translations efficiency calculations

Usage

```
fpkm_calc(counts, lengthSize, librarySize)
```

Arguments

counts	a list, # of read hits per group
lengthSize	a list of lengths per group
librarySize	a numeric of size 1, the # of reads in library

Value

a numeric vector

References

doi: 10.1038/nbt.1621

See Also

Other features: computeFeaturesCage, computeFeatures, disengagementScore, distToCds, distToTSS, entropy, floss, fpkm, fractionLength, initiationScore, insideOutsideORF, isInFrame, isOverlapping, kozakSequenceScore, orfScore, rankOrder, ribosomeReleaseScore, ribosomeStallingScore, startRegionCoverage, startRegion, subsetCoverage, translationalEff

---

fractionLength *Fraction Length*

Description

Fraction Length is defined as

```
(widths of grl)/tx_len
```

so that each group in the grl is divided by the corresponding transcript.

Usage

```
fractionLength(grl, tx_len)
```

Arguments

grl	a GRangesList object with usually either leaders, cds', 3' utrs or ORFs. ORFs are a special case, see argument tx_len
tx_len	the transcript lengths of the transcripts, a named (tx names) vector of integers. If you have the transcripts as GRangesList, call 'ORFik:::widthPerGroup(tx, TRUE)'. If you used CageSeq to reannotate leaders, then the tss for the the leaders have changed, therefore the tx lengths have changed. To account for that call: 'tx_len <- widthPerGroup(extendLeaders(tx, cageFiveUTRs))' and calculate fraction length using 'fractionLength(grl, tx_len)'.

Value

a numeric vector of ratios

References

doi: 10.1242/dev.098343

See Also

Other features: computeFeaturesCage, computeFeatures, disengagementScore, distToCds, distToTSS, entropy, floss, fpkm_calc, fpkm, initiationScore, insideOutsideORF, isInFrame, isOverlapping, kozakSequenceScore, orfScore, rankOrder, ribosomeReleaseScore, ribosomeStallingScore, startRegionCoverage, startRegion, subsetCoverage, translationalEff

Examples

```
ORF <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(1, 10, 20), end = c(5, 15, 25)),
               strand = "+")
grl <- GRangesList(tx1_1 = ORF)
# grl must have same names as cds + _1 etc, so that they can be matched.
tx <-  GRangesList(tx1 = GRanges("1", IRanges(1, 50), "+"))
fractionLength(grl, ORFik:::widthPerGroup(tx, keep.names = TRUE))
```

---

fread.bed *Load bed file as GRanges.*

Description

Wraps around rtracklayer::import.bed and tries to speed up loading with the use of data.table. Supports gzip, gz, bgz and bed formats. Also safer chromosome naming with the argument chrStyle

Usage

```
fread.bed(filePath, chrStyle = NULL)
```

Arguments

filePath	The location of the bed file
chrStyle	a GRanges object, or a character style (Default: NULL) to get seqlevelsStyle from. Is chromosome 1 called chr1 or 1, is mitocondrial chromosome called MT or chrM etc. Will use 1st seqlevel- style if more are present. Like: c("NCBI", "UCSC") -> pick "NCBI"

Value

a GRanges object

See Also

Other utils: bedToGR, convertToOneBasedRanges, findFa, optimizeReads, readBam, readWig

Examples

```
# path to example CageSeq data from hg19 heart sample
cageData <- system.file("extdata", "cage-seq-heart.bed.bgz",
                        package = "ORFik")
fread.bed(cageData)
```

---

gcContent *Get GC content*

Description

0.5 means 50

Usage

```
gcContent(seqs, fa)
```

Arguments

seqs	a character vector of ranges, or ranges as GRangesList
fa	fasta index file .fai file, either path to it, or the loaded FaFile, default (NULL), only set if you give ranges as GRangesList

Value

a numeric vector of gc content scores

Examples

```
# Usually the ORFs are found in orfik, which makes names for you etc.
# Here we make an example from scratch
seqName <- "Chromosome"
ORF1 <- GRanges(seqnames = seqName,
                  ranges = IRanges(c(1007, 1096), width = 60),
                  strand = c("+", "+"))
ORF2 <- GRanges(seqnames = seqName,
                   ranges = IRanges(c(400, 100), width = 30),
                   strand = c("-", "-"))
ORFs <- GRangesList(tx1 = ORF1, tx2 = ORF2)
ORFs <- makeORFNames(ORFs) # need ORF names
# get path to FaFile for sequences
faFile <- system.file("extdata", "genome.fasta", package = "ORFik")
gcContent(ORFs, faFile)
```

getNGenesCoverage *Get number of genes per grouping*

Description

Get number of genes per grouping

Usage

```
getNGenesCoverage(coverage)
```

Arguments

coverage	a data.table with coverage

Value

number of genes in coverage

groupGRangesBy *Group GRanges*

Description

It will group / split the GRanges object by the argument ‘other‘. For example if you would like to to group GRanges object by gene, set other to gene names.

Usage

```
groupGRangesBy(gr, other = NULL)
```

Arguments

gr	a GRanges object
other	a vector of unique names to group by (default: NULL)

Details

If ‘other‘ is not specified function will try to use the names of the GRanges object. It will then be similar to ‘split(gr, names(gr))‘.

It is important that all groups in ‘other‘ are unique, otherwise duplicates will be grouped together.

Value

a GRangesList named after names(Granges) if other is NULL, else names are from unique(other)

Examples

```
ORFranges <- GRanges(seqnames = Rle(rep("1", 3)),
                     ranges = IRanges(start = c(1, 10, 20),
                                      end = c(5, 15, 25)),
                     strand = "+")
ORFranges2 <- GRanges("1",
                      ranges = IRanges(start = c(20, 30, 40),
                                       end = c(25, 35, 45)),
                      strand = "+")
names(ORFranges) = rep("tx1_1", 3)
names(ORFranges2) = rep("tx1_2", 3)
grl <- GRangesList(tx1_1 = ORFranges, tx1_2 = ORFranges2)
gr <- unlist(grl, use.names = FALSE)
## now recreate the grl
## group by orf
grltest <- groupGRangesBy(gr) # using the names to group
identical(grl, grltest) ## they are identical

## group by transcript
names(gr) <- txNames(gr)
grltest <- groupGRangesBy(gr)
identical(grl, grltest) ## they are not identical
```

`groupings` *Get number of ranges per group as an iterator*

Description

Get number of ranges per group as an iterator

Usage

```
groupings(grl)
```

Arguments

`grl`	GRangesList

Value

an integer vector

Examples

```
grl <- GRangesList(GRanges("1", c(1, 3, 5), "+"),
                   GRanges("1", c(19, 21, 23), "+"))
ORFik:::groupings(grl)
```

gSort *Sort a GRangesList, helper.*

Description

A helper for [sortPerGroup()]. A faster, more versatile reimplementation of GenomicRanges::sort() Normally not used directly. Groups first each group, then either decreasing or increasing (on starts if byStarts == T, on ends if byStarts == F)

Usage

```
gSort(grl, decreasing = FALSE, byStarts = TRUE)
```

Arguments

grl	a GRangesList
decreasing	should the first in each group have max(start(group)) ->T or min-> default(F) ?
byStarts	a logical T, should it order by starts or ends F.

Value

an equally named GRangesList, where each group is sorted within group.

hasHits *Hits from reads*

Description

Finding GRanges groups that have overlap hits with reads Similar to

Usage

```
hasHits(grl, reads, keep.names = FALSE)
```

Arguments

grl	a GRanges or GRangesList
reads	a GAlignment or GRanges object with reads
keep.names	logical (F), keep names or not

Value

a list of logicals, T == hit, F == no hit

initiationScore *Get initiation score for a GRangesList of ORFs*

Description

initiationScore tries to check how much each TIS region resembles, the average of the CDS TIS regions.

Usage

```
initiationScore(grl, cds, tx, reads, pShifted = TRUE)
```

Arguments

grl	a GRangesList object with ORFs
cds	a GRangesList object with coding sequences
tx	a GrangesList of transcripts covering grl.
reads	ribosomal footprints, given as Galignment object or Granges
pShifted	a logical (TRUE), are riboseq reads p-shifted?

Details

Since this features uses a distance matrix for scoring, values are distributed like this: As result there is one value per ORF: 0.000: means that ORF had no reads -1.000: means that ORF is identical to average of CDS 1.000: means that orf is maximum different than average of CDS

Value

an integer vector, 1 score per ORF, with names of grl

References

doi: 10.1186/s12915-017-0416-0

See Also

Other features: computeFeaturesCage, computeFeatures, disengagementScore, distToCds, distToTSS, entropy, floss, fpkm_calc, fpkm, fractionLength, insideOutsideORF, isInFrame, isOverlapping, kozakSequenceScore, orfScore, rankOrder, ribosomeReleaseScore, ribosomeStallingScore, startRegionCoverage, startRegion, subsetCoverage, translationalEff

Examples

```
# Good hiting ORF
ORF <- GRanges(seqnames = "1",
               ranges = IRanges(21, 40),
               strand = "+")
names(ORF) <- c("tx1")
grl <- GRangesList(tx1 = ORF)
# 1 width p-shifted reads
reads <- GRanges("1", IRanges(c(21, 23, 50, 50, 50, 53, 53, 56, 59),
```

```
                              width = 1), "+")
score(reads) <- 28 # original width
cds <- GRanges(seqnames = "1",
               ranges = IRanges(50, 80),
               strand = "+")
cds <- GRangesList(tx1 = cds)
tx <- GRanges(seqnames = "1",
               ranges = IRanges(1, 85),
               strand = "+")
tx <- GRangesList(tx1 = tx)

initiationScore(grl, cds, tx, reads, pShifted = TRUE)
```

`insideOutsideORF` *Inside/Outside score (IO)*

Description

Inside/Outside score is defined as

`(reads over ORF)/(reads outside ORF and within transcript)`

A pseudo-count of one is added to both the ORF and outside sums.

Usage

```
insideOutsideORF(grl, RFP, GtfOrTx, ds = NULL, RFP.sorted = FALSE)
```

Arguments

`grl`	a `GRangesList` object with usually either leaders, cds', 3' utrs or ORFs
`RFP`	ribo seq reads as GAlignment, GRanges or GRangesList object
`GtfOrTx`	if Gtf: a TxDb object of a gtf file that transcripts will be extracted with 'exonsBy(Gtf, by = "tx", use.names = TRUE)', if a GrangesList will use as is
`ds`	numeric vector (NULL), disengagement score. If you have already calculated `disengagementScore`, input here to save time.
`RFP.sorted`	logical (F), have you ran this line: `RFP <-sort(RFP[countOverlaps(RFP,tx,type = "within") > 0])` Normally not touched, for internal optimization purposes.

Value

a named vector of numeric values of scores

References

doi: 10.1242/dev.098345

See Also

Other features: `computeFeaturesCage`, `computeFeatures`, `disengagementScore`, `distToCds`, `distToTSS`, `entropy`, `floss`, `fpkm_calc`, `fpkm`, `fractionLength`, `initiationScore`, `isInFrame`, `isOverlapping`, `kozakSequenceScore`, `orfScore`, `rankOrder`, `ribosomeReleaseScore`, `ribosomeStallingScore`, `startRegionCoverage`, `startRegion`, `subsetCoverage`, `translationalEff`

Examples

```
# Check inside outside score of a ORF within a transcript
ORF <- GRanges("1",
               ranges = IRanges(start = c(20, 30, 40),
                                end = c(25, 35, 45)),
               strand = "+")

grl <- GRangesList(tx1_1 = ORF)

tx1 <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(1, 10, 20, 30, 40, 50),
                                end = c(5, 15, 25, 35, 45, 200)),
               strand = "+")
tx <- GRangesList(tx1 = tx1)
RFP <- GRanges(seqnames = "1",
                  ranges = IRanges(start = c(1, 4, 30, 60, 80, 90),
                                   end = c(30, 33, 63, 90, 110, 120)),
                  strand = "+")

insideOutsideORF(grl, RFP, tx)
```

`is.grl` *Helper function to check for GRangesList*

Description

Helper function to check for GRangesList

Usage

```
is.grl(class)
```

Arguments

`class`	the class you want to check if is GRL, either a character from class or the object itself.

Value

a boolean

See Also

Other validity: `checkRFP`, `checkRNA`, `is.ORF`, `is.gr_or_grl`, `validGRL`, `validSeqlevels`

is.gr_or_grl *Helper function to check for GRangesList or GRanges class*

Description

Helper function to check for GRangesList or GRanges class

Usage

```
is.gr_or_grl(class)
```

Arguments

class	the class you want to check if is GRL or GR, either a character from class or the object itself.

Value

a boolean

See Also

Other validity: checkRFP, checkRNA, is.ORF, is.grl, validGRL, validSeqlevels

is.ORF *Check if all requirements for an ORFik ORF is accepted.*

Description

Check if all requirements for an ORFik ORF is accepted.

Usage

```
is.ORF(grl)
```

Arguments

grl	a GRangesList or GRanges to check

Value

a logical (TRUE/FALSE)

See Also

Other validity: checkRFP, checkRNA, is.gr_or_grl, is.grl, validGRL, validSeqlevels

`isInFrame` *Find frame for each orf relative to cds*

Description

Input of this function, is the output of the function [distToCds()], or any other relative ORF frame.

Usage

```
isInFrame(dists)
```

Arguments

`dists`	a vector of distances between ORF and cds

Details

possible outputs: 0: orf is in frame with cds 1: 1 shifted from cds 2: 2 shifted from cds

Value

a logical vector

References

doi: 10.1074/jbc.R116.733899

See Also

Other features: `computeFeaturesCage`, `computeFeatures`, `disengagementScore`, `distToCds`, `distToTSS`, `entropy`, `floss`, `fpkm_calc`, `fpkm`, `fractionLength`, `initiationScore`, `insideOutsideORF`, `isOverlapping`, `kozakSequenceScore`, `orfScore`, `rankOrder`, `ribosomeReleaseScore`, `ribosomeStallingScore`, `startRegionCoverage`, `startRegion`, `subsetCoverage`, `translationalEff`

Examples

```
# simple example
isInFrame(c(3,6,8,11,15))

# GRangesList example
grl <- GRangesList(tx1_1 = GRanges("1", IRanges(1,10), "+"))
fiveUTRs <- GRangesList(tx1 = GRanges("1", IRanges(1,20), "+"))
dist <- distToCds(grl, fiveUTRs)
isInFrame <- isInFrame(dist)
```

isOverlapping *Find frame for each orf relative to cds*

Description

Input of this function, is the output of the function [distToCds()]

Usage

```
isOverlapping(dists)
```

Arguments

dists	a vector of distances between ORF and cds

Value

a logical vector

References

doi: 10.1074/jbc.R116.733899

See Also

Other features: computeFeaturesCage, computeFeatures, disengagementScore, distToCds, distToTSS, entropy, floss, fpkm_calc, fpkm, fractionLength, initiationScore, insideOutsideORF, isInFrame, kozakSequenceScore, orfScore, rankOrder, ribosomeReleaseScore, ribosomeStallingScore, startRegionCoverage, startRegion, subsetCoverage, translationalEff

Examples

```
# simple example
isOverlapping(c(-3,-6,8,11,15))

# GRangesList example
grl <- GRangesList(tx1_1 = GRanges("1", IRanges(1,10), "+"))
fiveUTRs <- GRangesList(tx1 = GRanges("1", IRanges(1,20), "+"))
dist <- distToCds(grl, fiveUTRs)
isOverlapping <- isOverlapping(dist)
```

isPeriodic *Find if there is periodicity in the vector*

Description

Checks if there is a periodicity and if the periodicity is 3.

Usage

```
isPeriodic(x)
```

Arguments

x	(numeric) Vector of values to detect periodicity of 3 like in RiboSeq data.

Details

It uses Fourier transform for finding periodic vectors

Value

a logical, if it is periodic.

---

kozakHeatmap *Make sequence region heatmap relative to scoring*

Description

Given sequences, DNA or RNA. And some score, ribo-seq fpkm, TE etc. Create a heatmap divided per letter in seqs, by how strong the score is.

Usage

```
kozakHeatmap(seqs, rate, start, stop, center = ceiling((stop - start +
  1)/2), min.observations = ">q1", skip.startCodon = FALSE,
  xlab = "TIS", type = "ribo-seq")
```

Arguments

seqs	the sequences (character vector, DNAStringSet)
rate	a scoring vector (equal size to seqs)
start	position in seqs to start at (first is 1)
stop	position in seqs to stop at (first is 1)
center	position in seqs to center at (first is 1), center will be +1 in heatmap
min.observations	How many observations per position per letter to accept? numeric or quantile, default (">q1", bigger than quartile 1 (25 percentile)). You can do (10), to get all with more than 10 observations.
skip.startCodon	startCodon is defined as after centering (position 1, 2 and 3). Should they be skipped ? default (FALSE). Not relevant if you are not doing Translation initiation sites (TIS).
xlab	Region you are checking, default (TIS)
type	What type is the rate scoring ? default (ribo-seq)

Details

It will create blocks around best rate per position

Value

a ggplot of the heatmap

Examples

```
## Not run:
if (requireNamespace("BSgenome.Hsapiens.UCSC.hg19")) {
  txdbFile <- system.file("extdata", "hg19_knownGene_sample.sqlite",
                          package = "GenomicFeatures")
  #Extract sequences of Coding sequences.
  cds <- loadRegion(txdbFile, "cds")
  tx <- loadRegion(txdbFile, "mrna")

  # Get region to check
  kozakRegions <- startRegionString(cds, tx, BSgenome.Hsapiens.UCSC.hg19::Hsapiens
                                    , upstream = 4, 5)
  # Some toy ribo-seq fpkm scores on cds
  set.seed(3)
  fpkm <- sample(1:115, length(cds), replace = TRUE)
  kozakHeatmap(kozakRegions, fpkm, 1, 9, skip.startCodon = F)
}

## End(Not run)
```

`kozakSequenceScore` *Make a score for each ORFs start region by proximity to Kozak*

Description

The closer the sequence is to the Kozak sequence the higher the score, based on the experimental pwms from article referenced. Minimum score is 0 (worst correlation), max is 1 (the best base per column was chosen).

Usage

```
kozakSequenceScore(grl, tx, faFile, species = "human",
  include.N = FALSE)
```

Arguments

`grl`	a `GRangesList` grouped by ORF
`tx`	a `GRangesList`, the reference area for ORFs, each ORF must have a corresponding tx.
`faFile`	a FaFile from the fasta file, see ?FaFile. Can also be path to fastaFile with fai file in same dir.
`species`	("human"), which species to use, currently supports human, zebrafish and mouse (m. musculus). You can also specify a pfm for your own species. Syntax of pfm is an rectangular integer matrix, where all columns must sum to the same value, normally 100. See example for more information. Rows are in order: c("A", "C", "G", "T")
`include.N`	logical (F), if TRUE, allow N bases to be counted as hits, score will be average of the other bases. If True, N bases will be added to pfm, automaticly, so dont include them if you make your own pfm.

Details

Ranges that does not have minimum 15 length (the kozak requirement as a sliding window of size 15 around grl start), will be set to score 0. Since they should not have the posibility to make a ribosome binding.

Value

a numeric vector with values between 0 and 1

an integer vector, one score per orf

References

doi: https://doi.org/10.1371/journal.pone.0108475

See Also

Other features: computeFeaturesCage, computeFeatures, disengagementScore, distToCds, distToTSS, entropy, floss, fpkm_calc, fpkm, fractionLength, initiationScore, insideOutsideORF, isInFrame, isOverlapping, orfScore, rankOrder, ribosomeReleaseScore, ribosomeStallingScore, startRegionCoverage, startRegion, subsetCoverage, translationalEff

Examples

```
# Usually the ORFs are found in orfik, which makes names for you etc.
# Here we make an example from scratch
seqName <- "Chromosome"
ORF1 <- GRanges(seqnames = seqName,
                  ranges = IRanges(c(1007, 1096), width = 60),
                  strand = c("+", "+"))
ORF2 <- GRanges(seqnames = seqName,
                   ranges = IRanges(c(400, 100), width = 30),
                   strand = c("-", "-"))
ORFs <- GRangesList(tx1 = ORF1, tx2 = ORF2)
ORFs <- makeORFNames(ORFs) # need ORF names
tx <- extendLeaders(ORFs, 100)
# get faFile for sequences
faFile <- FaFile(system.file("extdata", "genome.fasta", package = "ORFik"))
kozakSequenceScore(ORFs, tx, faFile)
# For more details see vignettes.
```

---

lastExonEndPerGroup *Get last end per granges group*

Description

Get last end per granges group

Usage

```
lastExonEndPerGroup(grl, keep.names = TRUE)
```

Arguments

grl	a GRangesList
keep.names	a boolean, keep names or not

Value

a Rle(keep.names = T), or integer vector(F)

Examples

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
                   ranges = IRanges(c(7, 14), width = 3),
                   strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
                    ranges = IRanges(c(4, 1), c(9, 3)),
                    strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
lastExonEndPerGroup(grl)
```

---

lastExonPerGroup *Get last exon per GRangesList group*

Description

grl must be sorted, call ORFik:::sortPerGroup if needed

Usage

```
lastExonPerGroup(grl)
```

Arguments

grl	a GRangesList

Value

a GRangesList of the last exon per group

Examples

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
                   ranges = IRanges(c(7, 14), width = 3),
                   strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
                    ranges = IRanges(c(4, 1), c(9, 3)),
                    strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
lastExonPerGroup(grl)
```

lastExonStartPerGroup *Get last start per granges group*

Description

Get last start per granges group

Usage

```
lastExonStartPerGroup(grl, keep.names = TRUE)
```

Arguments

grl	a GRangesList
keep.names	a boolean, keep names or not

Value

a Rle(keep.names = T), or integer vector(F)

Examples

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
                   ranges = IRanges(c(7, 14), width = 3),
                   strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
                    ranges = IRanges(c(4, 1), c(9, 3)),
                    strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
lastExonStartPerGroup(grl)
```

libraryTypes *Which type of experiments?*

Description

Which type of experiments?

Usage

```
libraryTypes(df)
```

Arguments

df	an ORFik experiment data.frame

Value

library types (character vector)

loadRegion *Load transcript region*

Description

Load GRangesList if input is not already GRangesList.

Usage

```
loadRegion(txdb, part = "tx")
```

Arguments

txdb	a TxDb file or a path to one of: (.gtf ,.gff, .gff2, .gff2, .db or .sqlite), if it is a GRangesList, it will return it self.
part	a character, one of: tx, leader, cds, trailer, intron, mrna NOTE: difference between tx and mrna is that tx are all transcripts, while mrna are all transcripts with a cds

Value

a GrangesList of region

Examples

```
gtf <- system.file("extdata", "annotations.gtf", package = "ORFik")
loadRegion(gtf, "intron")
```

loadRegions *Get all regions of transcripts specified*

Description

Get all regions of transcripts specified

Usage

```
loadRegions(txdb, parts = c("mrna", "leaders", "cds", "trailers"),
  extension = "", envir = .GlobalEnv)
```

Arguments

txdb	a TxDb file, an ORFik experiment or a path to one of: (.gtf ,.gff, .gff2, .gff2, .db or .sqlite)
parts	the transcript parts you want
extension	What to add on the name after leader, like: B -> leadersB
envir	Which environment to save to, default (.GlobalEnv)

Value

NULL (regions set by envir assignment)

loadTranscriptType *Load transcripts of given biotype*

Description

Like rRNA, snoRNA etc. NOTE: Only works on gtf/gff, not .db object for now. Also note that these anotations are not perfect, some rRNA annotations only contain 5S rRNA etc. If your gtf does not contain evertyhing you need, use a resource like repeatmasker and download a gtf: https://genome.ucsc.edu/cgi-bin/hgTables

Usage

```
loadTranscriptType(path, part = "rRNA", tx = NULL)
```

Arguments

path	path to gtf/gff
part	a character, default rRNA. Can also be: snoRNA, tRNA etc. As long as that biotype is defined in the gtf.
tx	a GRangesList of transcripts (Optional, default NULL), add to save run time.

Value

a GRangesList of transcript of that type

References

doi: 10.1002/0471250953.bi0410s25

loadTxdb *General loader for txdb*

Description

Useful to allow fast TxDb loader like .db

Usage

```
loadTxdb(txdb, chrStyle = NULL)
```

Arguments

txdb	a TxDb file, an ORFik experiment or a path to one of: (.gtf ,.gff, .gff2, .gff2, .db or .sqlite)
chrStyle	a GRanges object, or a character style (Default: NULL) to get seqlevelsStyle from. Is chromosome 1 called chr1 or 1, is mitocondrial chromosome called MT or chrM etc. Will use 1st seqlevel- style if more are present. Like: c("NCBI", "UCSC") -> pick "NCBI"

Value

a TxDb object

Examples

```
library(GenomicFeatures)
# Get the gtf txdb file
txdbFile <- system.file("extdata", "hg19_knownGene_sample.sqlite",
                        package = "GenomicFeatures")
txdb <- loadDb(txdbFile)
```

---

longestORFs *Get longest ORF per stop site*

Description

Rule: if seqname, strand and stop site is equal, take longest one. Else keep. If IRangesList or IRanges, seqnames are groups, if GRanges or GRangesList seqnames are the seqlevels (e.g. chromosomes/transcripts)

Usage

```
longestORFs(grl)
```

Arguments

grl	a GRangesList/IRangesList, GRanges/IRanges of ORFs

Value

a GRangesList/IRangesList, GRanges/IRanges (same as input)

See Also

Other ORFHelpers: defineTrailer, mapToGRanges, orfID, startCodons, startSites, stopCodons, stopSites, txNames, uniqueGroups, uniqueOrder

Examples

```
ORF1 = GRanges("1", IRanges(10,21), "+")
ORF2 = GRanges("1", IRanges(1,21), "+") # <- longest
grl <- GRangesList(ORF1 = ORF1, ORF2 = ORF2)
longestORFs(grl) # get only longest
```

makeExonRanks *Make grouping for exon structures.*

Description

Either by transcript or by original groupings. Must be ordered, so that same transcripts are ordered together.

Usage

```
makeExonRanks(grl, byTranscript = FALSE)
```

Arguments

grl	a GRangesList
byTranscript	if ORfs are by transcript, check duplicates

Value

an integer vector of indices for exon ranks

makeORFNames *Make ORF names per orf*

Description

grl must be grouped by transcript If a list of orfs are grouped by transcripts, but does not have ORF names, then create them and return the new GRangesList

Usage

```
makeORFNames(grl, groupByTx = TRUE)
```

Arguments

grl	a GRangesList
groupByTx	logical (T), should output GRangesList be grouped by transcripts (T) or by ORFs (F)?

Value

(GRangesList) with ORF names, grouped by transcripts, sorted.

Examples

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
                   ranges = IRanges(c(7, 14), width = 3),
                   strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
                    ranges = IRanges(c(4, 1), c(9, 3)),
                    strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
makeORFNames(grl)
```

---

mapToGRanges *Map orfs to genomic coordinates*

Description

Creates GRangesList from the results of ORFs_as_List and the GRangesList used to find the ORFs

Usage

```
mapToGRanges(grl, result, groupByTx = TRUE)
```

Arguments

grl	A GRangesList of the original sequences that gave the orfs in Genomic coordinates.
result	IRangesList A list of the results of finding uorfs list syntax is: Per list group in IRangesList is per grl index. In transcript coordinates. The names are grl index as character.
groupByTx	logical (T), should output GRangesList be grouped by transcripts (T) or by ORFs (F)?

Details

There is no check on invalid matches, so be carefull if you use this function directly.

Value

A GRangesList of ORFs.

See Also

Other ORFHelpers: defineTrailer, longestORFs, orfID, startCodons, startSites, stopCodons, stopSites, txNames, uniqueGroups, uniqueOrder

matchColors *Match coloring of coverage plot*

Description

Check that colors match with the number of fractions.

Usage

```
matchColors(coverage, colors)
```

Arguments

coverage	a data.table with coverage
colors	a character vector of colors

Value

number of genes in coverage

matchNaming *Match naming of GRangesList*

Description

Given a GRangesList and a reference, make the naming convention and the number of metacolumns equal to reference

Usage

```
matchNaming(gr, reference)
```

Arguments

gr	a GRangesList or GRanges object
reference	a GRangesList of a reference

Value

a GRangesList

matchSeqStyle *A wrapper for seqlevelsStyle*

Description

To make sure chromosome naming is correct (chr1 vs 1 vs I etc)

Usage

```
matchSeqStyle(range, chrStyle = NULL)
```

Arguments

range	a ranged object, (GRanges, GAlignment etc)
chrStyle	a GRanges object, or a character style (Default: NULL) to get seqlevelsStyle from. Is chromosome 1 called chr1 or 1, is mitocondrial chromosome called MT or chrM etc. Will use 1st seqlevel- style if more are present. Like: c("NCBI", "UCSC") -> pick "NCBI"

Value

a GAlignment/GRanges object depending on input.

metaWindow *Calculate meta-coverage of reads around input GRanges/List object.*

Description

Sums up coverage over set of GRanges objects as a meta representation.

Usage

```
metaWindow(x, windows, scoring = "sum", withFrames = FALSE,
  zeroPosition = NULL, scaleTo = 100, fraction = NULL,
  feature = NULL, forceUniqueEven = !is.null(scoring))
```

Arguments

x	GRangesList/GRanges object of your reads. Remember to resize them beforehand to width of 1 to focus on 5' ends of footprints, if that is wanted.
windows	GRangesList or GRanges of your ranges
scoring	a character, one of (zscore, transcriptNormalized, mean, median, sum, sumLength, NULL), see ?coverageScorings
withFrames	a logical (TRUE), return positions with the 3 frames, relative to zeroPosition. zeroPosition is frame 0.
zeroPosition	an integer DEFAULT (NULL), the point if all windows are equal size, that should be set to position 0. Like leaders and cds combination, then 0 is the TIS and -1 is last base in leader. NOTE!: if windows have different widths, this will be ignored.

scaleTo	an integer (100), if windows have different size, a meta window can not directly be created, since a meta window must have equal size for all windows. Rescale all windows to scaleTo. i.e c(1,2,3) -> size 2 -> c(1, sum(2,3)) etc.
fraction	a character/integer (NULL), the fraction i.e (27) for read length 27, or ("LSU") for large sub-unit TCP-seq.
feature	a character string, info on region. Usually either gene name, transcript part like cds, leader, or CpG motifs etc.
forceUniqueEven,	a logical (TRUE), if TRUE; require that all windows are of same width and even. To avoid bugs. FALSE if score is NULL.

Value

A data.table with scored counts (score) of reads mapped to positions (position) specified in windows along with frame (frame).

See Also

Other coverage: coverageScorings, scaledWindowPositions, windowPerReadLength

Examples

```
library(GenomicRanges)
windows <- GRangesList(GRanges("chr1", IRanges(c(50, 100), c(80, 200)),
                               "-"))
x <- GenomicRanges::GRanges(
  seqnames = "chr1",
  ranges =  IRanges::IRanges(c(100, 180), c(200, 300)),
  strand = "-")
metaWindow(x, windows, withFrames = FALSE)
```

---

nrow,experiment-method

Internal nrow function for ORFik experiment

Description

Internal nrow function for ORFik experiment

Usage

```
## S4 method for signature 'experiment'
nrow(x)
```

Arguments

x	an ORFik experiment

Value

number of rows in experiment (integer)

numCodons *Get number of codons*

Description

Choose only whole codons, or with stubs. But usually there are no ORFs that are 17 bases etc.

Usage

```
numCodons(grl, as.integer = TRUE, keep.names = FALSE)
```

Arguments

grl	a GRangesList object
as.integer	a logical (TRUE), remove stub codons
keep.names	a logical (FALSE)

Value

an integer vector

numExonsPerGroup *Get list of the number of exons per group*

Description

Can also be used generaly to get number of GRanges object per GRangesList group

Usage

```
numExonsPerGroup(grl, keep.names = TRUE)
```

Arguments

grl	a GRangesList
keep.names	a boolean, keep names or not

Value

an integer vector of counts

Examples

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
                   ranges = IRanges(c(7, 14), width = 3),
                   strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
                    ranges = IRanges(c(4, 1), c(9, 3)),
                    strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
numExonsPerGroup(grl)
```

optimizeReads *Find optimized subset of valid reads*

Description

If more than a million reads, keep only the ones that overlap within the grl ranges.

Usage

```
optimizeReads(grl, reads)
```

Arguments

grl	a GRangesList or GRanges object
reads	a GRanges or GAlignment object

Value

the reads as GRanges or GAlignment

See Also

Other utils: bedToGR, convertToOneBasedRanges, findFa, fread.bed, readBam, readWig

orfID *Get id's for each orf*

Description

These id's can be uniqued by isoform etc, this is not supported by GenomicRanges.

Usage

```
orfID(grl, with.tx = FALSE)
```

Arguments

grl	a GRangesList
with.tx	a boolean, include transcript names, if you want unique orfs, so that they dont have multiple versions on different isoforms, set it to FALSE.

Value

a character vector of ids, 1 per orf

See Also

Other ORFHelpers: defineTrailer, longestORFs, mapToGRanges, startCodons, startSites, stopCodons, stopSites, txNames, uniqueGroups, uniqueOrder

orfScore *Get ORFscore for a GRangesList of ORFs*

Description

ORFscore tries to check whether the first frame of the 3 possible frames in an ORF has more reads than second and third frame.

Usage

```
orfScore(grl, RFP, is.sorted = FALSE)
```

Arguments

grl	a GRangesList object with ORFs
RFP	ribosomal footprints, given as Galignment object, Granges or GRangesList
is.sorted	logical (F), is grl sorted.

Details

Pseudocode: assume rff - is reads fraction in specific frame

```
ORFScore = log(rrf1 + rrf2 + rrf3)
```

For all ORFs where rrf2 or rrf3 is bigger than rff1, negate the resulting value.

```
ORFScore[rrf1Smaller] <- ORFScore[rrf1Smaller] * -1
```

As result there is one value per ORF: Positive values say that the first frame have the most reads, negative values say that the first frame does not have the most reads. NOTE: If reads are not of size 1, then a read from 1-4 on range of 1-4, will get scores frame1 = 2, frame2 = 1, frame3 = 1. What could be logical is that only the 5' end is important, so that only frame1 = 1, to get this, you first resize reads to 5'end only.

NOTE: p shifting is not exact, so many ORFs will get a bad ORF score.

Value

a data.table with 4 columns, the orfscore (ORFScores) and score of each of the 3 tiles (frame_zero_RP, frame_one_RP, frame_two_RP)

References

doi: 10.1002/embj.201488411

See Also

Other features: computeFeaturesCage, computeFeatures, disengagementScore, distToCds, distToTSS, entropy, floss, fpkm_calc, fpkm, fractionLength, initiationScore, insideOutsideORF, isInFrame, isOverlapping, kozakSequenceScore, rankOrder, ribosomeReleaseScore, ribosomeStallingScore, startRegionCoverage, startRegion, subsetCoverage, translationalEff

Examples

```
ORF <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(1, 10, 20), end = c(5, 15, 25)),
               strand = "+")
names(ORF) <- c("tx1", "tx1", "tx1")
grl <- GRangesList(tx1_1 = ORF)
RFP <- GRanges("1", IRanges(25, 25), "+") # 1 width position based
score(RFP) <- 28 # original width
orfScore(grl, RFP) # negative because more hits on frames 1,2 than 0.

# example with positive result, more hits on frame 0 (in frame of ORF)
RFP <- GRanges("1", IRanges(c(1, 1, 1, 25), width = 1), "+")
score(RFP) <- c(28, 29, 31, 28) # original width
orfScore(grl, RFP)
```

---

outputLibs *Output bam/bed/wig files to R as variables*

Description

Variable names defined by df

Usage

```
outputLibs(df, chrStyle = NULL, envir = .GlobalEnv)
```

Arguments

df	an ORFik experiment data.frame
chrStyle	the sequencelevels style (GRanges object or chr)
envir	environment to save to, default (.GlobalEnv)

Value

NULL (libraries set by envir assignment)

---

overlapsToCoverage *Get overlaps and convert to coverage list*

Description

Get overlaps and convert to coverage list

Usage

```
overlapsToCoverage(gr, reads, keep.names = TRUE, type = "any")
```

Arguments

gr	a GRanges object, to get coverage of.
reads	a GAlignment or GRanges object of RiboSeq, RnaSeq etc.
keep.names	logical (T), keep names or not.
type	a string (any), argument for countOverlaps.

Value

a Rle, one list per group with # of hits per position.

See Also

Other ExtendGenomicRanges: asTX, coveragePerTiling, reduceKeepAttr, tile1, txSeqsFromFa, windowPerGroup

Examples

```
ORF <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(1, 10, 20),
                                end = c(5, 15, 25)),
               strand = "+")
names(ORF) <- "tx1"
reads <- GRanges("1", IRanges(25, 25), "+")
overlapsToCoverage(ORF, reads)
```

---

parseCigar *Shift ribo-seq reads using cigar string*

Description

Shift ribo-seq reads using cigar string

Usage

```
parseCigar(cigar, shift, is_plus_strand)
```

Arguments

cigar	the cigar of the reads
shift	the shift as integer
is_plus_strand	logical

Value

the shifted read

pmapFromTranscriptF *Faster pmapFromTranscript*

Description

This version tries to fix the shortcommings of GenomicFeature's version. Much faster and uses less memory. Implemented as dynamic program optimized c++ code.

Usage

```
pmapFromTranscriptF(x, transcripts, removeEmpty = FALSE)
```

Arguments

x	IRangesList/IRanges/GRanges to map to genomic coordinates
transcripts	a GRangesList to map against
removeEmpty	a logical, remove non hit exons, else they are set to 0.

Details

The length of x must be the same as length of transcripts. Only exception is if x have integer names like (1, 3, 3, 5), so that x[1] maps to 1, x[2] maps to transcript 3 etc.

Value

a GRangesList of mapped reads, names from ranges are kept.

Examples

```
ranges <- IRanges(start = c( 5, 6), end = c(10, 10))
seqnames = rep("chr1", 2)
strands = rep("-", 2)
grl <- split(GRanges(seqnames, IRanges(c(85, 70), c(89, 82)), strands),
             c(1, 1))
ranges <- split(ranges, c(1,1)) # both should be mapped to transcript 1
pmapFromTranscriptF(ranges, grl, TRUE)
```

prettyScoring *Prettify scoring name*

Description

Prettify scoring name

Usage

```
prettyScoring(scoring)
```

Arguments

scoring	a character (the scoring)

Value

a new scoring name or the same if pretty

---

pSitePlot *Plot area around TIS as histogram*

Description

Usefull to validate p-shifting is correct Can be used for any coverage of region around a point, like TIS, TSS, stop site etc.

Usage

```
pSitePlot(hitMap, length = 29, region = "start", output = NULL,
  type = "canonical CDS", scoring = "Averaged counts",
  forHeatmap = FALSE)
```

Arguments

hitMap	a data.frame/data.table, given from metaWindow (must have columns: position, (score or count) and frame)
length	an integer (29), which length is this for?
region	a character (start), either "start or "stop"
output	character (NULL), if set, saves the plot as pdf or png to path given. If no format is given, is save as pdf.
type	character (canonical CDS), type for plot
scoring	character (Average sum) which scoring did you use ?
forHeatmap	a logical (FALSE), should the plot be part of a heatmap? It will scale it differently. Removing x and y labels, and truncate spaces between bars.

Details

The region is represented as a histogram with different colors for the 3 frames. To make it easy to see patterns in the reads. Remember if you want to change anything like colors, just return the ggplot object, and reassign like: obj + scale_color_brewer() etc.

Value

a ggplot object of the coverage plot, NULL if output is set, then the plot will only be saved to location.

See Also

Other coveragePlot: coverageHeatMap, savePlot, windowCoveragePlot

Examples

```
# An ORF
grl <- GRangesList(tx1 = GRanges("1", IRanges(1, 6), "+"))
# Ribo-seq reads
range <- IRanges(c(rep(1, 3), 2, 3, rep(4, 2), 5, 6), width = 1 )
reads <- GRanges("1", range, "+")
coverage <- coveragePerTiling(grl, reads, TRUE, as.data.table = TRUE,
                              withFrames = TRUE)
pSitePlot(coverage)

# See vignette for more examples
```

`rankOrder` *ORF rank in transcripts*

Description

Creates an ordering of ORFs per transcript, so that ORF with the most upstream start codon is 1, second most upstream start codon is 2, etc. Must input a grl made from ORFik, txNames_2 -> 2.

Usage

```
rankOrder(grl)
```

Arguments

`grl`	a `GRangesList` object with ORFs

Value

a numeric vector of integers

References

doi: 10.1074/jbc.R116.733899

See Also

Other features: `computeFeaturesCage`, `computeFeatures`, `disengagementScore`, `distToCds`, `distToTSS`, `entropy`, `floss`, `fpkm_calc`, `fpkm`, `fractionLength`, `initiationScore`, `insideOutsideORF`, `isInFrame`, `isOverlapping`, `kozakSequenceScore`, `orfScore`, `ribosomeReleaseScore`, `ribosomeStallingScore`, `startRegionCoverage`, `startRegion`, `subsetCoverage`, `translationalEff`

Examples

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
                   ranges = IRanges(c(7, 14), width = 3),
                   strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
                    ranges = IRanges(c(4, 1), c(9, 3)),
                    strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
```

```
grl <- ORFik:::makeORFNames(grl)
rankOrder(grl)
```

`read.experiment` *Read ORFik experiment*

Description

An object to massivly simplify your coding, it is similar to systempipeR's 'target' table. By containing filepaths and info for each library in some experiment.

Usage

```
read.experiment(file)
```

Arguments

`file`	a .csv file following ORFik experiment style, or a template data.frame from create.experiment()

Details

Simplest way to make is to call create.experiment on some folder with libraries and see what you get. Some of the fields might be needed to fill in manually. The important thing is that each row must be unique (excluding filepath), that means if it has replicates then that must be said explicit. And all filepaths must be unique and have files with size > 0. Syntax: libtype (library type): rna-seq, ribo-seq, CAGE etc. rep (replicate): 1,2,3 etc condition: WT (wild-type), control, target, mzdicer, starved etc. fraction: 18, 19 (fractinations), or other ways to split library. filepath: Full filepath to file

The file must be csv and be a valid ORFik experiment

Value

an ORFik experiment

Examples

```
# From file
## Not run:
df <- read.experiment(filepath) # <- valid .csv file

## End(Not run)
# From (create.experiment() template)
template <- create.experiment(dir = system.file("extdata", "", package = "ORFik"),
                              exper = "ORFik", txdb = system.file("extdata",
                                    "annotations.gtf",
                                    package = "ORFik"),
                              viewTemplate = FALSE)
template$X5[6] <- "heart" # <- fix non unique row
# read experiment
df <- read.experiment(template)
```

readBam *Custom bam reader*

Description

Only for single end reads Safer version that handles the most important error done. In the future will use a faster .bam loader for big .bam files in R.

Usage

```
readBam(path, chrStyle = NULL)
```

Arguments

path	a character path to .bam file
chrStyle	a GRanges object, or a character style (Default: NULL) to get seqlevelsStyle from. Is chromosome 1 called chr1 or 1, is mitocondrial chromosome called MT or chrM etc. Will use 1st seqlevel- style if more are present. Like: c("NCBI", "UCSC") -> pick "NCBI"

Value

a GAlignment object of bam file

See Also

Other utils: bedToGR, convertToOneBasedRanges, findFa, fread.bed, optimizeReads, readWig

readWidths *Get read widths*

Description

Input any reads, e.g. ribo-seq object and get width of reads, this is to avoid confusion between width, qwidth and meta column containing original read width.

Usage

```
readWidths(reads, after.softclips = TRUE)
```

Arguments

reads	a GRanges or GAlignment object.
after.softclips	logical (TRUE), include softclips in width

Details

If input is p-shifted and GRanges, the "$size" or "$score" colum" must exist, and the column must contain the original read widths. In ORFik "$size" have higher priority than "$score" for defining length. ORFik P-shifting creates a $size column, other softwares like shoelaces creates a score column.

Remember to think about how you define length. Like the question: is a Illumina error mismatch sufficient to reduce size of read and how do you know what is biological variance and what are Illumina errors?

Value

an integer vector of widths

Examples

```
gr <- GRanges("chr1", 1)
readWidths(gr)

# GAlignment with hit (1M) and soft clipped base (1S)
ga <- GAlignments(seqnames = "1", pos = as.integer(1), cigar = "1M1S",
 strand = factor("+", levels = c("+", "-", "*")))
readWidths(ga) # Without soft-clip bases

readWidths(ga, after.softclips = FALSE) # With soft-clip bases
```

---

readWig *Custom wig reader*

Description

Given 2 wig files, first is forward second is reverse. Merge them and return as GRanges object. If they contain name reverse and forward, first and second order does not matter, it will search for forward and reverse.

Usage

```
readWig(path, chrStyle = NULL)
```

Arguments

path	a character path to .bam file
chrStyle	a GRanges object, or a character style (Default: NULL) to get seqlevelsStyle from. Is chromosome 1 called chr1 or 1, is mitocondrial chromosome called MT or chrM etc. Will use 1st seqlevel- style if more are present. Like: c("NCBI", "UCSC") -> pick "NCBI"

Value

a GAlignment object of bam file

See Also

Other utils: bedToGR, convertToOneBasedRanges, findFa, fread.bed, optimizeReads, readBam

---

`reassignTSSbyCage` *Reassign all Transcript Start Sites (TSS)*

Description

Given a GRangesList of 5' UTRs or transcripts, reassign the start sites using max peaks from CageSeq data. A max peak is defined as new TSS if it is within boundary of 5' leader range, specified by 'extension' in bp. A max peak must also be higher than minimum CageSeq peak cutoff specified in 'filterValue'. The new TSS will then be the positioned where the cage read (with highest read count in the interval). If removeUnused is TRUE, leaders without cage hits, will be removed, if FALSE the original TSS will be used.

Usage

```
reassignTSSbyCage(fiveUTRs, cage, extension = 1000, filterValue = 1,
  restrictUpstreamToTx = FALSE, removeUnused = FALSE,
  preCleanup = TRUE, cageMcol = FALSE)
```

Arguments

`fiveUTRs`	(GRangesList) The 5' leaders or full transcript sequences
`cage`	Either a filePath for the CageSeq file as .bed .bam or .wig, with possible compressions (".gzip", ".gz", ".bgz"), or already loaded CageSeq peak data as GRanges or GAlignment. NOTE: If it is a .bam file, it will add a score column by running: convertToOneBasedRanges(cage, method = "5prime", addScoreColumn = TRUE)
`extension`	The maximum number of basses upstream of the TSS to search for CageSeq peak.
`filterValue`	The minimum number of reads on cage position, for it to be counted as possible new tss. (represented in score column in CageSeq data) If you already filtered, set it to 0.
`restrictUpstreamToTx`	a logical (FALSE). If TRUE: restrict leaders to not extend closer than 5 bases from closest upstream leader, set this to TRUE.
`removeUnused`	logical (FALSE), if False: (standard is to set them to original annotation), If TRUE: remove leaders that did not have any cage support.
`preCleanup`	logical (TRUE), if TRUE, remove all reads in region (-5:-1, 1:5) of all original tss in leaders. This is to keep original TSS if it is only +/- 5 bases from the original.
`cageMcol`	a logical (FALSE), if TRUE, add a meta column to the returned object with the raw CAGE counts in support for new TSS.

Details

Note: If you used CAGEr, you will get reads of a probability region, with always score of 1. Remember then to set filterValue to 0. And you should use the 5' end of the read as input, use: ORFik:::convertToOneBasedRanges(cage) NOTE on filtervalue: To get high quality TSS, set filtervalue to median count of reads overlapping per leader. This will make you discard a lot of new TSS positions though. I usually use 10 as a good standard.

TIP: do summary(countOverlaps(fiveUTRs, cage)) so you can find a good cutoff value for noise.

Value

a GRangesList of newly assigned TSS for fiveUTRs, using CageSeq data.

See Also

Other CAGE: assignTSSByCage, reassignTxDbByCage

Examples

```
# example 5' leader, notice exon_rank column
fiveUTRs <- GenomicRanges::GRangesList(
  GenomicRanges::GRanges(seqnames = "chr1",
                         ranges = IRanges::IRanges(1000, 2000),
                         strand = "+",
                         exon_rank = 1))
names(fiveUTRs) <- "tx1"

# make fake CageSeq data from promoter of 5' leaders, notice score column
cage <- GenomicRanges::GRanges(
  seqnames = "1",
  ranges =  IRanges::IRanges(500, width = 1),
  strand = "+",
  score = 10) # <- Number of tags (reads) per position
# notice also that seqnames use different naming, this is fixed by ORFik
# finally reassign TSS for fiveUTRs
reassignTSSbyCage(fiveUTRs, cage)
# See vignette for example using gtf file and real CAGE data.
```

---

reassignTxDbByCage *Input a txdb and reassign the TSS for each transcript by CAGE*

Description

Given a TxDb object, reassign the start site per transcript using max peaks from CageSeq data. A max peak is defined as new TSS if it is within boundary of 5' leader range, specified by 'extension' in bp. A max peak must also be higher than minimum CageSeq peak cutoff specified in 'filterValue'. The new TSS will then be the positioned where the cage read (with highest read count in the interval).

Usage

```
reassignTxDbByCage(txdb, cage, extension = 1000, filterValue = 1,
  restrictUpstreamToTx = FALSE, removeUnused = FALSE,
  preCleanup = TRUE)
```

Arguments

txdb	a TxDb file, an ORFik experiment or a path to one of: (.gtf ,.gff, .gff2, .gff2, .db or .sqlite)
cage	Either a filePath for the CageSeq file as .bed .bam or .wig, with possible compressions (".gzip", ".gz", ".bgz"), or already loaded CageSeq peak data as GRanges or GAlignment. NOTE: If it is a .bam file, it will add a score column by running: convertToOneBasedRanges(cage, method = "5prime", addScoreColumn = TRUE)
extension	The maximum number of basses upstream of the TSS to search for CageSeq peak.
filterValue	The minimum number of reads on cage position, for it to be counted as possible new tss. (represented in score column in CageSeq data) If you already filtered, set it to 0.
restrictUpstreamToTx	a logical (FALSE). If TRUE: restrict leaders to not extend closer than 5 bases from closest upstream leader, set this to TRUE.
removeUnused	logical (FALSE), if False: (standard is to set them to original annotation), If TRUE: remove leaders that did not have any cage support.
preCleanup	logical (TRUE), if TRUE, remove all reads in region (-5:-1, 1:5) of all original tss in leaders. This is to keep original TSS if it is only +/- 5 bases from the original.

Details

Note: If you used CAGEr, you will get reads of a probability region, with always score of 1. Remember then to set filterValue to 0. And you should use the 5' end of the read as input, use: ORFik:::convertToOneBasedRanges(cage)

Value

a TxDb obect of reassigned transcripts

See Also

Other CAGE: assignTSSByCage, reassignTSSbyCage

Examples

```
## Not run:
library(GenomicFeatures)
# Get the gtf txdb file
txdbFile <- system.file("extdata", "hg19_knownGene_sample.sqlite",
package = "GenomicFeatures")
cagePath <- system.file("extdata", "cage-seq-heart.bed.bgz",
package = "ORFik")
reassignTxDbByCage(txdbFile, cagePath)

## End(Not run)
```

reduceKeepAttr *Reduce GRanges / GRangesList*

Description

Extends function reduce by trying to keep names and meta columns, if it is a GRangesList. It also does not lose sorting for GRangesList, since original reduce sorts all by ascending. If keep.names == FALSE, it's just the normal GenomicRanges::reduce with sorting negative strands descending for GRangesList.

Usage

```
reduceKeepAttr(grl, keep.names = FALSE, drop.empty.ranges = FALSE,
  min.gapwidth = 1L, with.revmap = FALSE,
  with.inframe.attrib = FALSE, ignore.strand = FALSE)
```

Arguments

grl	a GRangesList or GRanges object
keep.names	(FALSE) keep the names and meta columns of the GRangesList
drop.empty.ranges	(FALSE) if a group is empty (width 0), delete it.
min.gapwidth	(1L) how long gap can it be to say they belong together
with.revmap	(FALSE) return info on which mapped to which
with.inframe.attrib	(FALSE) For internal use.
ignore.strand	(FALSE), can different strands be reduced together.

Value

A reduced GRangesList

See Also

Other ExtendGenomicRanges: asTX, coveragePerTiling, overlapsToCoverage, tile1, txSeqsFromFa, windowPerGroup

Examples

```
ORF <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(1, 2, 3), end = c(1, 2, 3)),
               strand = "+")
# For GRanges
reduceKeepAttr(ORF, keep.names = TRUE)
# For GRangesList
grl <- GRangesList(tx1_1 = ORF)
reduceKeepAttr(grl, keep.names = TRUE)
```

remakeTxdbExonIds *Get new exon ids*

Description

Get new exon ids

Usage

```
remakeTxdbExonIds(txList)
```

Arguments

txList	a list, call of as.list(txdb)

Value

a new valid ordered list of exon ids (integer)

removeMetaCols *Removes meta columns*

Description

Removes meta columns

Usage

```
removeMetaCols(grl)
```

Arguments

grl	a GRangesList or GRanges object

Value

same type and structure as input without meta columns

`removeORFsWithinCDS` *Remove ORFs that are within cds*

Description

Remove ORFs that are within cds

Usage

```
removeORFsWithinCDS(grl, cds)
```

Arguments

`grl`	(GRangesList), the ORFs to filter
`cds`	(GRangesList), the coding sequences (main ORFs on transcripts), to filter against.

Value

(GRangesList) of filtered uORFs

See Also

Other uorfs: `addCdsOnLeaderEnds`, `filterUORFs`, `removeORFsWithSameStartAsCDS`, `removeORFsWithSameStopAsCD`
`removeORFsWithStartInsideCDS`, `uORFSearchSpace`

`removeORFsWithSameStartAsCDS`
Remove ORFs that have same start site as the CDS

Description

Remove ORFs that have same start site as the CDS

Usage

```
removeORFsWithSameStartAsCDS(grl, cds)
```

Arguments

`grl`	(GRangesList), the ORFs to filter
`cds`	(GRangesList), the coding sequences (main ORFs on transcripts), to filter against.

Value

(GRangesList) of filtered uORFs

See Also

Other uorfs: `addCdsOnLeaderEnds`, `filterUORFs`, `removeORFsWithSameStopAsCDS`, `removeORFsWithStartInsideCD`
`removeORFsWithinCDS`, `uORFSearchSpace`

`removeORFsWithSameStopAsCDS` *Remove ORFs that have same stop site as the CDS*

Description

Remove ORFs that have same stop site as the CDS

Usage

```
removeORFsWithSameStopAsCDS(grl, cds)
```

Arguments

`grl`	(GRangesList), the ORFs to filter
`cds`	(GRangesList), the coding sequences (main ORFs on transcripts), to filter against.

Value

(GRangesList) of filtered uORFs

See Also

Other uorfs: `addCdsOnLeaderEnds`, `filterUORFs`, `removeORFsWithSameStartAsCDS`, `removeORFsWithStartInsideC`
`removeORFsWithinCDS`, `uORFSearchSpace`

`removeORFsWithStartInsideCDS` *Remove ORFs that have start site within the CDS*

Description

Remove ORFs that have start site within the CDS

Usage

```
removeORFsWithStartInsideCDS(grl, cds)
```

Arguments

`grl`	(GRangesList), the ORFs to filter
`cds`	(GRangesList), the coding sequences (main ORFs on transcripts), to filter against.

Value

(GRangesList) of filtered uORFs

See Also

Other uorfs: `addCdsOnLeaderEnds`, `filterUORFs`, `removeORFsWithSameStartAsCDS`, `removeORFsWithSameStopAsCD`
`removeORFsWithinCDS`, `uORFSearchSpace`

removeTxdbExons *Remove exons in txList that are not in fiveUTRs*

Description

Remove exons in txList that are not in fiveUTRs

Usage

```
removeTxdbExons(txList, fiveUTRs)
```

Arguments

txList	a list, call of as.list(txdb)
fiveUTRs	a GRangesList of 5' leaders

Value

a list, modified call of as.list(txdb)

removeTxdbTranscripts *Remove specific transcripts in txdb List*

Description

Remove all transcripts, except the ones in fiveUTRs.

Usage

```
removeTxdbTranscripts(txList, fiveUTRs)
```

Arguments

txList	a list, call of as.list(txdb)
fiveUTRs	a GRangesList of 5' leaders

Value

a txList

`restrictTSSByUpstreamLeader` *Restrict extension of 5' UTRs to closest upstream leader end*

Description

Basicly this function restricts all startSites, to the upstream GRangesList objects end. Usually leaders, for CAGE. Example: leader1: start on 10, leader2: stop on 8, extend leader1 to 5 -> this function will resize leader1 to 9, to be outside leader2, so that CAGE reads can not wrongly overlap.

Usage

```
restrictTSSByUpstreamLeader(fiveUTRs, shiftedfiveUTRs)
```

Arguments

`fiveUTRs`	The 5' leader sequences as GRangesList
`shiftedfiveUTRs`	The 5' leader sequences as GRangesList shifted by CAGE

Value

GRangesList object of restricted fiveUTRs

`ribosomeReleaseScore` *Ribosome Release Score (RRS)*

Description

Ribosome Release Score is defined as

```
(RPFs over ORF)/(RPFs over 3' utrs)
```

and additionaly normalized by lengths. If RNA is added as argument, it will normalize by RNA counts to justify location of 3' utrs. It can be understood as a ribosome stalling feature. A pseudo-count of one was added to both the ORF and downstream sums.

Usage

```
ribosomeReleaseScore(grl, RFP, GtfOrThreeUtrs, RNA = NULL)
```

Arguments

`grl`	a `GRangesList` object with usually either leaders, cds', 3' utrs or ORFs.
`RFP`	RiboSeq reads as GAlignment, GRanges or GRangesList object
`GtfOrThreeUtrs`	if Gtf: a TxDb object of a gtf file transcripts is called from: 'threeUTRsByTranscript(Gtf, use.names = TRUE)', if object is GRangesList, it is presumed to be the 3' utrs
`RNA`	RnaSeq reads as GAlignment, GRanges or GRangesList object

Value

a named vector of numeric values of scores, NA means that no 3' utr was found for that transcript.

References

doi: 10.1016/j.cell.2013.06.009

See Also

Other features: computeFeaturesCage, computeFeatures, disengagementScore, distToCds, distToTSS, entropy, floss, fpkm_calc, fpkm, fractionLength, initiationScore, insideOutsideORF, isInFrame, isOverlapping, kozakSequenceScore, orfScore, rankOrder, ribosomeStallingScore, startRegionCoverage, startRegion, subsetCoverage, translationalEff

Examples

```
ORF <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(1, 10, 20), end = c(5, 15, 25)),
               strand = "+")
grl <- GRangesList(tx1_1 = ORF)
threeUTRs <- GRangesList(tx1 = GRanges("1", IRanges(40, 50), "+"))
RFP <- GRanges("1", IRanges(25, 25), "+")
RNA <- GRanges("1", IRanges(1, 50), "+")
ribosomeReleaseScore(grl, RFP, threeUTRs, RNA)
```

---

ribosomeStallingScore *Ribosome Stalling Score (RSS)*

Description

Is defined as

```
(RPFs over ORF stop sites)/(RPFs over ORFs)
```

and normalized by lengths A pseudo-count of one was added to both the ORF and downstream sums.

Usage

```
ribosomeStallingScore(grl, RFP)
```

Arguments

grl	a GRangesList object with usually either leaders, cds', 3' utrs or ORFs.
RFP	RiboSeq reads as GAlignment, GRanges or GRangesList object

Value

a named vector of numeric values of RSS scores

References

doi: 10.1016/j.cels.2017.08.004

See Also

Other features: computeFeaturesCage, computeFeatures, disengagementScore, distToCds, distToTSS, entropy, floss, fpkm_calc, fpkm, fractionLength, initiationScore, insideOutsideORF, isInFrame, isOverlapping, kozakSequenceScore, orfScore, rankOrder, ribosomeReleaseScore, startRegionCoverage, startRegion, subsetCoverage, translationalEff

Examples

```
ORF <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(1, 10, 20), end = c(5, 15, 25)),
               strand = "+")
grl <- GRangesList(tx1_1 = ORF)
RFP <- GRanges("1", IRanges(25, 25), "+")
ribosomeStallingScore(grl, RFP)
```

---

save.experiment *Save experiment to disc*

Description

Save experiment to disc

Usage

```
save.experiment(df, file)
```

Arguments

df	an ORFik experiment data.frame
file	name of file to save df as

Value

NULL (experiment save only)

savePlot *Helper function for writing plots to disc*

Description

Helper function for writing plots to disc

Usage

```
savePlot(plot, output = NULL, width = 200, height = 150, dpi = 300)
```

Arguments

plot	the ggplot to save
output	character string (NULL), if set, saves the plot as pdf or png to path given. If no format is given, is save as png.
width	width of output in mm
height	height of output in mm
dpi	(300) dpi of plot

Value

a ggplot object of the coverage plot, NULL if output is set, then the plot will only be saved to location.

See Also

Other coveragePlot: coverageHeatMap, pSitePlot, windowCoveragePlot

scaledWindowPositions *Scale windows to a meta window of size*

Description

For example scale a coverage plot of a all human CDS to width 100

Usage

```
scaledWindowPositions(grl, reads, scaleTo = 100, scoring = "meanPos")
```

Arguments

grl	GRangesList or GRanges of your ranges
reads	GRanges object of your reads.
scaleTo	an integer (100), if windows have different size, a meta window can not directly be created, since a meta window must have equal size for all windows. Rescale all windows to scaleTo. i.e c(1,2,3) -> size 2 -> c(1, mean(2,3)) etc. Can also be a vector, 1 number per grl group.
scoring	a character, one of (meanPos, sumPos)

Details

Nice for making metaplots, the score will be mean of merged positions.

Value

A data.table with scored counts (counts) of reads mapped to positions (position) specified in windows along with frame (frame).

See Also

Other coverage: coverageScorings, metaWindow, windowPerReadLength

Examples

```
library(GenomicRanges)
windows <- GRangesList(GRanges("chr1", IRanges(1, 200), "-"))
x <- GenomicRanges::GRanges(
  seqnames = "chr1",
  ranges =  IRanges::IRanges(c(1, 100, 199), c(2, 101, 200)),
  strand = "-")
scaledWindowPositions(windows, x, scaleTo = 100)
```

---

seqnamesPerGroup *Get list of seqnames per granges group*

Description

Get list of seqnames per granges group

Usage

```
seqnamesPerGroup(grl, keep.names = TRUE)
```

Arguments

grl	a GRangesList
keep.names	a boolean, keep names or not

Value

a character vector or Rle of seqnames(if seqnames == T)

Examples

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
                   ranges = IRanges(c(7, 14), width = 3),
                   strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
                    ranges = IRanges(c(4, 1), c(9, 3)),
                    strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
seqnamesPerGroup(grl)
```

`shiftFootprints` *Shift footprints by selected offsets*

Description

Function shifts footprints (GRanges) using specified offsets for every of the specified lengths. Reads that do not conform to the specified lengths are filtered out and rejected. Reads are resized to single base in 5' end fashion, treated as p site. This function takes account for junctions in cigars of the reads. Length of the footprint is saved in size' parameter of GRanges output. Footprints are also sorted according to their genomic position, ready to be saved as a bed file.

Usage

```
shiftFootprints(footprints, shifts)
```

Arguments

`footprints`	(GAlignments) object of RiboSeq reads
`shifts`	a data.frame / data.table with minimum 2 columns, selected_lengths and selected_shifts. Output from `detectRibosomeShifts`

Details

The two columns in shift are: - fraction Numeric vector of lengths of footprints you select for shifting. - offsets_start Numeric vector of shifts for coresponding selected_lengths. eg. c(10, -10) with selected_lengths of c(31, 32) means length of 31 will be shifted left by 10. Footprints of length 32 will be shifted right by 10.

NOTE: It will remove softclips from valid width, the CIGAR 3S30M is qwidth 33, but will remove 3S so final read width is 30 in ORFik.

Value

A GRanges object of shifted footprints, sorted and resized to 1bp of p-site, with metacolumn "size" indicating footprint size before shifting and resizing, sorted in increasing order.

See Also

Other pshifting: `changePointAnalysis`, `detectRibosomeShifts`

Examples

```
## Not run:
# input path to gtf, or load it as TxDb.
gtf_file <- system.file("extdata", "annotations.gtf", package = "ORFik")
# load reads
riboSeq_file <- system.file("extdata", "ribo-seq.bam", package = "ORFik")
footprints <- GenomicAlignments::readGAlignments(
  riboSeq_file, param = ScanBamParam(flag = scanBamFlag(
    isDuplicate = FALSE, isSecondaryAlignment = FALSE)))
# detect the shifts automagically
shifts <- detectRibosomeShifts(footprints, gtf_file)
# shift the RiboSeq footprints
```

```
shiftedReads <- shiftFootprints(footprints, shifts)

## End(Not run)
```

`show,experiment-method` *experiment show definition*

Description

Show a simplified version of experiment.

Usage

```
## S4 method for signature 'experiment'
show(object)
```

Arguments

`object`	an ORFik experiment

Value

print state of experiment

`sortPerGroup` *Sort a GRangesList*

Description

A faster, more versatile reimplementation of `sort.GenomicRanges` for GRangesList, needed since the original works poorly for more than 10k groups. This function sorts each group, where "+" strands are increasing by starts and "-" strands are decreasing by ends.

Usage

```
sortPerGroup(grl, ignore.strand = FALSE)
```

Arguments

`grl`	a `GRangesList`
`ignore.strand`	a boolean, if FALSE: should minus strands be sorted from highest to lowest ends. If TRUE: from lowest to highest ends.

Details

Note: will not work if groups have equal names.

Value

an equally named GRangesList, where each group is sorted within group.

Examples

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
                   ranges = IRanges(c(14, 7), width = 3),
                   strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
                    ranges = IRanges(c(1, 4), c(3, 9)),
                    strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
sortPerGroup(grl)
```

---

splitIn3Tx *Create coverage of transcripts, split into the 3 parts.*

Description

The 3 parts of transcripts are the leaders, the cds' and trailers.

Usage

```
splitIn3Tx(leaders, cds, trailers, reads, windowSize = 100,
  fraction = "1")
```

Arguments

leaders	a GRangesList of leaders (5' UTRs)
cds	a GRangesList of coding sequences
trailers	a GRangesList of trailers (3' UTRs)
reads	GRanges or GAlignment of reads
windowSize	an integer (100), size of windows
fraction	a character (1), info on reads (which read length, or which type (RNA seq))

Value

a data.table with columns position, score

---

startCodons *Get the Start codons(3 bases) from a GRangesList of orfs grouped by orfs*

Description

In ATGTTTTGA, get the positions ATG. It takes care of exons boundaries, with exons < 3 length.

Usage

```
startCodons(grl, is.sorted = FALSE)
```

Arguments

grl	a GRangesList object
is.sorted	a boolean, a speedup if you know the ranges are sorted

Value

a GRangesList of start codons, since they might be split on exons

See Also

Other ORFHelpers: defineTrailer, longestORFs, mapToGRanges, orfID, startSites, stopCodons, stopSites, txNames, uniqueGroups, uniqueOrder

Examples

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
                   ranges = IRanges(c(7, 14), width = 3),
                   strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
                    ranges = IRanges(c(4, 1), c(9, 3)),
                    strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
startCodons(grl, is.sorted = FALSE)
```

---

startDefinition *Returns start codon definitions*

Description

According to: <http://www.ncbi.nlm.nih.gov/Taxonomy/taxonomyhome.html/ index.cgi?chapter=tgencodes#SG1> ncbi genetic code number for translation. This version is a cleaned up version, unknown indices removed.

Usage

```
startDefinition(transl_table)
```

Arguments

transl_table	numeric. NCBI genetic code number for translation.

Value

A string of START sies separatd with "|".

See Also

Other findORFs: findMapORFs, findORFsFasta, findORFs, findUORFs, stopDefinition

Examples

```
startDefinition
startDefinition(1)
```

`startRegion` *Start region as GRangesList*

Description

Get the start region of each ORF. If you want the start codon only, set upstream = 0 or just use `startCodons`. Standard is 2 upstream and 2 downstream, a width 5 window centered at start site. since p-shifting is not 100 usually the reads from the start site.

Usage

```
startRegion(grl, tx = NULL, is.sorted = TRUE, upstream = 2L,
  downstream = 2L)
```

Arguments

`grl`	a `GRangesList` object with usually either leaders, cds', 3' utrs or ORFs
`tx`	default NULL, a GRangesList of transcripts or (container region), names of tx must contain all grl names. The names of grl can also be the ORFik orf names. that is "txName_id"
`is.sorted`	logical (TRUE), is grl sorted.
`upstream`	an integer (2), relative region to get upstream from.
`downstream`	an integer (2), relative region to get downstream from

Details

If tx is null, then upstream will be forced to 0 and downstream to a maximum of grl width. Since there is no reference for splicing.

Value

a GRanges, or GRangesList object if any group had > 1 exon.

See Also

Other features: `computeFeaturesCage`, `computeFeatures`, `disengagementScore`, `distToCds`, `distToTSS`, `entropy`, `floss`, `fpkm_calc`, `fpkm`, `fractionLength`, `initiationScore`, `insideOutsideORF`, `isInFrame`, `isOverlapping`, `kozakSequenceScore`, `orfScore`, `rankOrder`, `ribosomeReleaseScore`, `ribosomeStallingScore`, `startRegionCoverage`, `subsetCoverage`, `translationalEff`

`startRegionCoverage` *Start region coverage*

Description

Get the number of reads in the start region of each ORF. If you want the start codon coverage only, set upstream = 0. Standard is 2 upstream and 2 downstream, a width 5 window centered at start site. since p-shifting is not 100 start site.

Usage

```
startRegionCoverage(grl, RFP, tx = NULL, is.sorted = TRUE,
  upstream = 2L, downstream = 2L)
```

Arguments

`grl`	a `GRangesList` object with usually either leaders, cds', 3' utrs or ORFs
`RFP`	ribo seq reads as GAlignment, GRanges or GRangesList object
`tx`	default NULL, a GRangesList of transcripts or (container region), names of tx must contain all grl names. The names of grl can also be the ORFik orf names. that is "txName_id"
`is.sorted`	logical (TRUE), is grl sorted.
`upstream`	an integer (2), relative region to get upstream from.
`downstream`	an integer (2), relative region to get downstream from

Details

If tx is null, then upstream will be force to 0 and downstream to a maximum of grl width. Since there is no reference for splicing.

Value

a numeric vector of counts

See Also

Other features: `computeFeaturesCage`, `computeFeatures`, `disengagementScore`, `distToCds`, `distToTSS`, `entropy`, `floss`, `fpkm_calc`, `fpkm`, `fractionLength`, `initiationScore`, `insideOutsideORF`, `isInFrame`, `isOverlapping`, `kozakSequenceScore`, `orfScore`, `rankOrder`, `ribosomeReleaseScore`, `ribosomeStallingScore`, `startRegion`, `subsetCoverage`, `translationalEff`

startRegionString *Get start region as DNA-strings per GRanges group*

Description

One window per start site, if upstream and downstream are both 0, then only the startsite is returned.

Usage

```
startRegionString(grl, tx, faFile, upstream = 20, downstream = 20)
```

Arguments

grl	a GRangesList of ranges to find regions in.
tx	a GRangesList of transcripts or (container region), names of tx must contain all gr names. The names of gr can also be the ORFik orf names. that is "txName_id"
faFile	a FaFile from the fasta file, see ?FaFile. Can also be path to fastaFile with fai file in same dir.
upstream	an integer (0), relative region to get upstream from.
downstream	an integer (0), relative region to get downstream from

Value

a character vector of start regions

startSites *Get the start sites from a GRangesList of orfs grouped by orfs*

Description

In ATGTTTTGG, get the position of the A.

Usage

```
startSites(grl, asGR = FALSE, keep.names = FALSE, is.sorted = FALSE)
```

Arguments

grl	a GRangesList object
asGR	a boolean, return as GRanges object
keep.names	a logical (FALSE), keep names of input.
is.sorted	a speedup, if you know the ranges are sorted

Value

if asGR is False, a vector, if True a GRanges object

See Also

Other ORFHelpers: defineTrailer, longestORFs, mapToGRanges, orfID, startCodons, stopCodons, stopSites, txNames, uniqueGroups, uniqueOrder

Examples

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
                   ranges = IRanges(c(7, 14), width = 3),
                   strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
                    ranges = IRanges(c(4, 1), c(9, 3)),
                    strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
startSites(grl, is.sorted = FALSE)
```

---

stopCodons *Get the Stop codons (3 bases) from a GRangesList of orfs grouped by orfs*

Description

In ATGTTTTGA, get the positions TGA. It takes care of exons boundaries, with exons < 3 length.

Usage

```
stopCodons(grl, is.sorted = FALSE)
```

Arguments

grl	a GRangesList object
is.sorted	a boolean, a speedup if you know the ranges are sorted

Value

a GRangesList of stop codons, since they might be split on exons

See Also

Other ORFHelpers: defineTrailer, longestORFs, mapToGRanges, orfID, startCodons, startSites, stopSites, txNames, uniqueGroups, uniqueOrder

Examples

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
                   ranges = IRanges(c(7, 14), width = 3),
                   strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
                    ranges = IRanges(c(4, 1), c(9, 3)),
                    strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
stopCodons(grl, is.sorted = FALSE)
```

stopDefinition *Returns stop codon definitions*

Description

According to: <http://www.ncbi.nlm.nih.gov/Taxonomy/taxonomyhome.html/ index.cgi?chapter=tgencodes#SG1> ncbi genetic code number for translation. This version is a cleaned up version, unknown indices removed.

Usage

```
stopDefinition(transl_table)
```

Arguments

transl_table	numeric. NCBI genetic code number for translation.

Value

A string of STOP sies separatd with "|".

See Also

Other findORFs: findMapORFs, findORFsFasta, findORFs, findUORFs, startDefinition

Examples

```
stopDefinition
stopDefinition(1)
```

stopSites *Get the stop sites from a GRangesList of orfs grouped by orfs*

Description

In ATGTTTTGC, get the position of the C.

Usage

```
stopSites(grl, asGR = FALSE, keep.names = FALSE, is.sorted = FALSE)
```

Arguments

grl	a GRangesList object
asGR	a boolean, return as GRanges object
keep.names	a logical (FALSE), keep names of input.
is.sorted	a speedup, if you know the ranges are sorted

Value

if asGR is False, a vector, if True a GRanges object

See Also

Other ORFHelpers: defineTrailer, longestORFs, mapToGRanges, orfID, startCodons, startSites, stopCodons, txNames, uniqueGroups, uniqueOrder

Examples

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
                   ranges = IRanges(c(7, 14), width = 3),
                   strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
                    ranges = IRanges(c(4, 1), c(9, 3)),
                    strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
stopSites(grl, is.sorted = FALSE)
```

---

strandBool *Get logical list of strands*

Description

Helper function to get a logical list of True/False, if GRangesList group have + strand = T, if - strand = F Also checks for * strands, so a good check for bugs

Usage

```
strandBool(grl)
```

Arguments

grl	a GRangesList or GRanges object

Value

a logical vector

Examples

```
gr <- GRanges(Rle(c("chr2", "chr2", "chr1", "chr3"), c(1, 3, 2, 4)),
              IRanges(1:10, width = 10:1),
              Rle(strand(c("-", "+", "*", "+", "-")), c(1, 2, 2, 3, 2)))
strandBool(gr)
```

strandPerGroup *Get list of strands per granges group*

Description

Get list of strands per granges group

Usage

```
strandPerGroup(grl, keep.names = TRUE)
```

Arguments

grl	a GRangesList
keep.names	a boolean, keep names or not

Value

a vector named/unnamed of characters

Examples

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
                   ranges = IRanges(c(7, 14), width = 3),
                   strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
                    ranges = IRanges(c(4, 1), c(9, 3)),
                    strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
strandPerGroup(grl)
```

subsetCoverage *Subset GRanges to get coverage.*

Description

GRanges object should be beforehand tiled to size of 1. This subsetting takes account for strand.

Usage

```
subsetCoverage(cov, y)
```

Arguments

cov	A coverage object from coverage()
y	GRanges object for which coverage should be extracted

Value

numeric vector of coverage of input GRanges object

See Also

Other features: computeFeaturesCage, computeFeatures, disengagementScore, distToCds, distToTSS, entropy, floss, fpkm_calc, fpkm, fractionLength, initiationScore, insideOutsideORF, isInFrame, isOverlapping, kozakSequenceScore, orfScore, rankOrder, ribosomeReleaseScore, ribosomeStallingScore, startRegionCoverage, startRegion, translationalEff

---

`subsetToFrame` *Subset GRanges to get desired frame. GRanges object should be beforehand tiled to size of 1. This subsetting takes account for strand.*

Description

Subset GRanges to get desired frame. GRanges object should be beforehand tiled to size of 1. This subsetting takes account for strand.

Usage

```
subsetToFrame(x, frame)
```

Arguments

x	A tiled to size of 1 GRanges object
frame	A numeric indicating which frame to extract

Value

GRanges object reduced to only first frame

`tile1` *Tile each GRangesList group to 1-base resolution.*

Description

Will tile a GRangesList into single bp resolution, each group of the list will be splited by positions of 1. Returned values are sorted as the same groups as the original GRangesList, except they are in bp resolutions. This is not supported originally by GenomicRanges.

Usage

```
tile1(grl, sort.on.return = TRUE, matchNaming = TRUE)
```

Arguments

grl	a GRangesList object with names
sort.on.return	logical (T), should the groups be sorted before return.
matchNaming	logical (T), should groups keep unlisted names and meta data.(This make the list very big, for > 100K groups)

Value

a GRangesList grouped by original group, tiled to 1

See Also

Other ExtendGenomicRanges: asTX, coveragePerTiling, overlapsToCoverage, reduceKeepAttr, txSeqsFromFa, windowPerGroup

Examples

```
gr1 <- GRanges("1", ranges = IRanges(start = c(1, 10, 20),
                                     end = c(5, 15, 25)),
               strand = "+")
gr2 <- GRanges("1", ranges = IRanges(start = c(20, 30, 40),
                                     end = c(25, 35, 45)),
               strand = "+")
names(gr1) = rep("tx1_1", 3)
names(gr2) = rep("tx1_2", 3)
grl <- GRangesList(tx1_1 = gr1, tx1_2 = gr2)
tile1(grl)
```

---

translationalEff *Translational efficiency*

Description

Uses RnaSeq and RiboSeq to get translational efficiency of every element in 'grl'. Translational efficiency is defined as:

```
(density of RPF within ORF) / (RNA expression of ORFs transcript)
```

Usage

```
translationalEff(grl, RNA, RFP, tx, with.fpkm = FALSE, pseudoCount = 0)
```

Arguments

grl	a GRangesList object can be either transcripts, 5' utrs, cds', 3' utrs or ORFs as a special case (uORFs, potential new cds' etc).
RNA	RnaSeq reads as GAlignment, GRanges or GRangesList object
RFP	RiboSeq reads as GAlignment, GRanges or GRangesList object
tx	a GRangesList of the transcripts. If you used cage data, then the tss for the the leaders have changed, therefor the tx lengths have changed. To account for that call: ' translationalEff(grl, RNA, RFP, tx = extendLeaders(tx, cageFiveUTRs)) ' where cageFiveUTRs are the reannotated by CageSeq data leaders.
with.fpkm	logical F, if true return the fpkm values together with translational efficiency
pseudoCount	an integer, 0, set it to 1 if you want to avoid NA and inf values. It also helps against bias from low depth libraries.

Value

a numeric vector of fpkm ratios, if with.fpkm is TRUE, return a data.table with te and fpkm values

References

doi: 10.1126/science.1168978

See Also

Other features: computeFeaturesCage, computeFeatures, disengagementScore, distToCds, distToTSS, entropy, floss, fpkm_calc, fpkm, fractionLength, initiationScore, insideOutsideORF, isInFrame, isOverlapping, kozakSequenceScore, orfScore, rankOrder, ribosomeReleaseScore, ribosomeStallingScore, startRegionCoverage, startRegion, subsetCoverage

Examples

```
ORF <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(1, 10, 20), end = c(5, 15, 25)),
               strand = "+")
grl <- GRangesList(tx1_1 = ORF)
RFP <- GRanges("1", IRanges(25, 25), "+")
RNA <- GRanges("1", IRanges(1, 50), "+")
tx <-  GRangesList(tx1 = GRanges("1", IRanges(1, 50), "+"))
# grl must have same names as cds + _1 etc, so that they can be matched.
te <- translationalEff(grl, RNA, RFP, tx, with.fpkm = TRUE, pseudoCount = 1)
te$fpkmRFP
te$te
```

---

txNames *Get transcript names from orf names*

Description

names must either be a column called names, or the names of the grl object. If it is already tx names, it returns the input

Usage

```
txNames(grl, unique = FALSE)
```

Arguments

grl	a GRangesList grouped by ORF or GRanges object
unique	a boolean, if true unique the names, used if several orfs map to same transcript and you only want the unique groups

Details

NOTE! Do not use _123 etc in end of transcript names if it is not ORFs. Else you will get errors. Just _ will work, but if transcripts are called ENST_123124124000 etc, it will crash, so substitute "_" with "." gsub("_", ".", names)

Value

a character vector of transcript names, without _* naming

See Also

Other ORFHelpers: defineTrailer, longestORFs, mapToGRanges, orfID, startCodons, startSites, stopCodons, stopSites, uniqueGroups, uniqueOrder

Examples

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
                   ranges = IRanges(c(7, 14), width = 3),
                   strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
                    ranges = IRanges(c(4, 1), c(9, 3)),
                    strand = c("-", "-"))
grl <- GRangesList(tx1_1 = gr_plus, tx2_1 = gr_minus)
# there are 2 orfs, both the first on each transcript
txNames(grl)
```

---

txNamesToGeneNames *Convert transcript names to gene names*

Description

Works for ensembl etc.

Usage

```
txNamesToGeneNames(txNames, txdb)
```

Arguments

txNames	character vector, the transcript names to convert.
txdb	the transcript database to use or gtf/gff path to it.

Value

character vector of gene names

txSeqsFromFa *Get transcript sequence from a GrangesList and a faFile or BSgenome*

Description

A small safety wrapper around `extractTranscriptSeqs` For debug of errors do: which(!(unique(seqnamesPerGroup(grl FALSE))

Usage

```
txSeqsFromFa(grl, faFile, is.sorted = FALSE)
```

Arguments

`grl`	a GRangesList object
`faFile`	FaFile, BSgenome or fasta/index file path used to find the transcripts
`is.sorted`	a speedup, if you know the ranges are sorted

Details

This happens usually when the grl contains chromsomes that the fasta file does not have. A normal error is that mitocondrial chromosome is called MT vs chrM even though they have same seqlevelsStyle. The above line will give you which chromosome it is missing.

Value

a DNAStringSet of the transcript sequences

See Also

Other ExtendGenomicRanges: `asTX`, `coveragePerTiling`, `overlapsToCoverage`, `reduceKeepAttr`, `tile1`, `windowPerGroup`

uniqueGroups *Get the unique set of groups in a GRangesList*

Description

Sometimes `GRangesList` groups might be identical, for example ORFs from different isoforms can have identical ranges. Use this function to reduce these groups to unique elements in `GRangesList` `grl`, without names and metacolumns.

Usage

```
uniqueGroups(grl)
```

Arguments

`grl`	a `GRangesList`

Value

a GRangesList of unique orfs

See Also

Other ORFHelpers: defineTrailer, longestORFs, mapToGRanges, orfID, startCodons, startSites, stopCodons, stopSites, txNames, uniqueOrder

Examples

```
gr1 <- GRanges("1", IRanges(1,10), "+")
gr2 <- GRanges("1", IRanges(20, 30), "+")
# make a grl with duplicated ORFs (gr1 twice)
grl <- GRangesList(tx1_1 = gr1, tx2_1 = gr2, tx3_1 = gr1)
uniqueGroups(grl)
```

---

uniqueOrder *Get unique ordering for GRangesList groups*

Description

This function can be used to calculate unique numerical identifiers for each of the GRangesList elements. Elements of GRangesList are unique when the GRanges inside are not duplicated, so ranges differences matter as well as sorting of the ranges.

Usage

```
uniqueOrder(grl)
```

Arguments

grl	a GRangesList

Value

an integer vector of indices of unique groups

See Also

uniqueGroups

Other ORFHelpers: defineTrailer, longestORFs, mapToGRanges, orfID, startCodons, startSites, stopCodons, stopSites, txNames, uniqueGroups

Examples

```
gr1 <- GRanges("1", IRanges(1,10), "+")
gr2 <- GRanges("1", IRanges(20, 30), "+")
# make a grl with duplicated ORFs (gr1 twice)
grl <- GRangesList(tx1_1 = gr1, tx2_1 = gr2, tx3_1 = gr1)
uniqueOrder(grl) # remember ordering

# example on unique ORFs
uniqueORFs <- uniqueGroups(grl)
# now the orfs are unique, let's map back to original set:
reMappedGrl <- uniqueORFs[uniqueOrder(grl)]
```

---

unlistGrl *Safe unlist*

Description

Same as [AnnotationDbi::unlist2()], keeps names correctly. Two differences is that if grl have no names, it will not make integer names, but keep them as null. Also if the GRangesList has names , and also the GRanges groups, then the GRanges group names will be kept.

Usage

```
unlistGrl(grl)
```

Arguments

grl	a GRangesList

Value

a GRanges object

Examples

```
ORF <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(1, 10, 20),
                                end = c(5, 15, 25)),
               strand = "+")
grl <- GRangesList(tx1_1 = ORF)
unlistGrl(grl)
```

uORFSearchSpace *Create search space to look for uORFs*

Description

Given a GRangesList of 5' UTRs or transcripts, reassign the start sites using max peaks from CageSeq data (if CAGE is given). A max peak is defined as new TSS if it is within boundary of 5' leader range, specified by 'extension' in bp. A max peak must also be higher than minimum CageSeq peak cutoff specified in 'filterValue'. The new TSS will then be the positioned where the cage read (with highest read count in the interval). If you want to include uORFs going into the CDS, add this argument too.

Usage

```
uORFSearchSpace(fiveUTRs, cage = NULL, extension = 1000,
  filterValue = 1, restrictUpstreamToTx = FALSE,
  removeUnused = FALSE, cds = NULL)
```

Arguments

fiveUTRs	(GRangesList) The 5' leaders or full transcript sequences
cage	Either a filePath for the CageSeq file as .bed .bam or .wig, with possible compressions (".gzip", ".gz", ".bgz"), or already loaded CageSeq peak data as GRanges or GAlignment. NOTE: If it is a .bam file, it will add a score column by running: convertToOneBasedRanges(cage, method = "5prime", addScoreColumn = TRUE)
extension	The maximum number of basses upstream of the TSS to search for CageSeq peak.
filterValue	The minimum number of reads on cage position, for it to be counted as possible new tss. (represented in score column in CageSeq data) If you already filtered, set it to 0.
restrictUpstreamToTx	a logical (FALSE). If TRUE: restrict leaders to not extend closer than 5 bases from closest upstream leader, set this to TRUE.
removeUnused	logical (FALSE), if False: (standard is to set them to original annotation), If TRUE: remove leaders that did not have any cage support.
cds	(GRangesList) CDS of relative fiveUTRs, applicable only if you want to extend 5' leaders downstream of CDS's, to allow upstream ORFs that can overlap into CDS's.

Value

a GRangesList of newly assigned TSS for fiveUTRs, using CageSeq data.

See Also

Other uorfs: addCdsOnLeaderEnds, filterUORFs, removeORFsWithSameStartAsCDS, removeORFsWithSameStopAsCD
removeORFsWithStartInsideCDS, removeORFsWithinCDS

Examples

```
# example 5' leader, notice exon_rank column
fiveUTRs <- GenomicRanges::GRangesList(
  GenomicRanges::GRanges(seqnames = "chr1",
                         ranges = IRanges::IRanges(1000, 2000),
                         strand = "+",
                         exon_rank = 1))
names(fiveUTRs) <- "tx1"

# make fake CageSeq data from promoter of 5' leaders, notice score column
cage <- GenomicRanges::GRanges(
  seqnames = "chr1",
  ranges =  IRanges::IRanges(500, 510),
  strand = "+",
  score = 10)

# finally reassign TSS for fiveUTRs
uORFSearchSpace(fiveUTRs, cage)
```

---

updateTxdbRanks *Update exon ranks of exon data.frame*

Description

Update exon ranks of exon data.frame

Usage

```
updateTxdbRanks(exons)
```

Arguments

exons	a data.frame, call of as.list(txdb)$splicings

Value

a data.frame, modified call of as.list(txdb)

---

updateTxdbStartSites *Update start sites of leaders*

Description

Update start sites of leaders

Usage

```
updateTxdbStartSites(txList, fiveUTRs, removeUnused)
```

Arguments

txList	a list, call of as.list(txdb)
fiveUTRs	a GRangesList of 5' leaders
removeUnused	logical (FALSE), remove leaders that did not have any cage support. (standard is to set them to original annotation)

Value

a list, modified call of as.list(txdb)

---

upstreamFromPerGroup *Get rest of objects upstream (inclusive)*

Description

Per group get the part upstream of position. upstreamFromPerGroup(tx, stopSites(fiveUTRs, asGR = TRUE)) will return the 5' utrs per transcript as GRangesList, usually used for interesting parts of the transcripts.

Usage

```
upstreamFromPerGroup(tx, upstreamFrom)
```

Arguments

tx	a GRangesList, usually of Transcripts to be changed
upstreamFrom	a vector of integers, for each group in tx, where is the new start point of first valid exon.

Details

If you don't want to include the points given in the region, use upstreamOfPerGroup

Value

a GRangesList of upstream part

See Also

Other GRanges: assignFirstExonsStartSite, assignLastExonsStopSite, downstreamFromPerGroup, downstreamOfPerGroup, upstreamOfPerGroup

upstreamOfPerGroup *Get rest of objects upstream (exclusive)*

Description

Per group get the part upstream of position upstreamOfPerGroup(tx, startSites(cds, asGR = TRUE)) will return the 5' utrs per transcript, usually used for interesting parts of the transcripts.

Usage

```
upstreamOfPerGroup(tx, upstreamOf, allowOutside = TRUE)
```

Arguments

tx	a GRangesList, usually of Transcripts to be changed
upstreamOf	a vector of integers, for each group in tx, where is the the base after the new stop point of last valid exon.
allowOutside	a logical (T), can upstreamOf extend outside range of tx, can set boundary as a false hit, so beware.

Value

a GRangesList of upstream part

See Also

Other GRanges: assignFirstExonsStartSite, assignLastExonsStopSite, downstreamFromPerGroup, downstreamOfPerGroup, upstreamFromPerGroup

validateExperiments *Validate ORFik experiment Check for valid non-empty files etc.*

Description

Validate ORFik experiment Check for valid non-empty files etc.

Usage

```
validateExperiments(df)
```

Arguments

df	an ORFik experiment data.frame

Value

NULL (Stops if failed)

validGRL *Helper Function to check valid GRangesList input*

Description

Helper Function to check valid GRangesList input

Usage

```
validGRL(class, type = "grl", checkNULL = FALSE)
```

Arguments

class	as character vector the given class of supposed GRangesList object
type	a character vector, is it gtf, cds, 5', 3', for messages.
checkNULL	should NULL classes be checked and return indeces of these?

Value

either NULL or indices (checkNULL == TRUE)

See Also

Other validity: checkRFP, checkRNA, is.ORF, is.gr_or_grl, is.grl, validSeqlevels

validSeqlevels *Helper function to find overlaping seqlevels*

Description

Useful to avoid warnings in bioC

Usage

```
validSeqlevels(grl, reads)
```

Arguments

grl	a GRangesList or GRanges object
reads	a GRanges or GAlignment object

Value

a character vector of valid seqlevels

See Also

Other validity: checkRFP, checkRNA, is.ORF, is.gr_or_grl, is.grl, validGRL

widthPerGroup *Get list of widths per granges group*

Description

Get list of widths per granges group

Usage

```
widthPerGroup(grl, keep.names = TRUE)
```

Arguments

grl	a GRangesList
keep.names	a boolean, keep names or not

Value

an integer vector (named/unnamed) of widths

Examples

```
gr_plus <- GRanges(seqnames = c("chr1", "chr1"),
                   ranges = IRanges(c(7, 14), width = 3),
                   strand = c("+", "+"))
gr_minus <- GRanges(seqnames = c("chr2", "chr2"),
                    ranges = IRanges(c(4, 1), c(9, 3)),
                    strand = c("-", "-"))
grl <- GRangesList(tx1 = gr_plus, tx2 = gr_minus)
widthPerGroup(grl)
```

windowCoveragePlot *Get meta coverage plot of reads*

Description

Spanning a region like a transcripts, plot how the reads distribute.

Usage

```
windowCoveragePlot(coverage, output = NULL, scoring = "zscore",
  colors = c("skyblue4", "orange"), title = "Coverage metaplot",
  type = "transcript", scaleEqual = FALSE)
```

Arguments

coverage	a data.table, e.g. output of scaledWindowCoverage
output	character string (NULL), if set, saves the plot as pdf or png to path given. If no format is given, is save as pdf.
scoring	character vector (zscore), either of zScore, transcriptNormalized, sum, mean, median, NULL. Set NULL if already scored.
colors	character vector colors to use in plot, will fix automaticly, using binary splits with colors c('skyblue4', 'orange').
title	a character (metaplot) (what is the title of plot?)
type	a character (transcript), what should legends say is the whole region? Transcript, gene, non coding rna etc.
scaleEqual	a logical (FALSE), should all fractions (rows), have same max value, for easy comparison of max values if needed.

Details

If coverage has a column called feature, this can be used to subdivide the meta coverage into parts as (5' UTRs, cds, 3' UTRs) These are the columns in the plot. The fraction column divide sequence libraries. Like ribo-seq and rna-seq. These are the rows of the plot. If you return this function without assigning it and output is NULL, it will automaticly plot the figure in your session. If output is assigned, no plot will be shown in session. NULL is returned and object is saved to output.

Colors: Remember if you want to change anything like colors, just return the ggplot object, and reassign like: obj + scale_color_brewer() etc.

Value

a ggplot object of the coverage plot, NULL if output is set, then the plot will only be saved to location.

See Also

Other coveragePlot: coverageHeatMap, pSitePlot, savePlot

Examples

```
library(data.table)
coverage <- data.table(position = seq(20),
                       score = sample(seq(20), 20, replace = TRUE))
windowCoveragePlot(coverage)

#Multiple plots in one frame:
coverage2 <- copy(coverage)
coverage$fraction <- "Ribo-seq"
coverage2$fraction <- "RNA-seq"
dt <- rbindlist(list(coverage, coverage2))
windowCoveragePlot(dt, scoring = "log10sum")

# See vignette for a more practical example
```

windowPerGroup *Get window region of GRanges object*

Description

If downstream is 20, it means the window will start 20 downstream of gr start site (-20 in relative transcript coordinates.) If upstream is 20, it means the window will start 20 upstream of gr start site (+20 in relative transcript coordinates.) It will keep exon structure of tx, so if -20 is on next exon, it jumps to next exon.

Usage

```
windowPerGroup(gr, tx, upstream = 0L, downstream = 0L)
```

Arguments

gr	a GRanges object (startSites and others, must be single point)
tx	a GRangesList of transcripts or (container region), names of tx must contain all gr names. The names of gr can also be the ORFik orf names. that is "txName_id"
upstream	an integer (0), relative region to get upstream from.
downstream	an integer (0), relative region to get downstream from

Details

If a region has a part that goes out of bounds, E.g if you try to get window around the CDS start site, goes longer than the 5' leader start site, it will set start to the edge boundary (the TSS of the transcript in this case). If region has no hit in bound, a width 0 GRanges object is returned. This is usefull for things like countOverlaps, since 0 hits will then always be returned for the correct object. If you don't want the 0 width windows, use `reduce()`

Value

a GRanges, or GRangesList object if any group had > 1 exon.

See Also

Other ExtendGenomicRanges: asTX, coveragePerTiling, overlapsToCoverage, reduceKeepAttr, tile1, txSeqsFromFa

Examples

```
# find 2nd codon of an ORF on a spliced transcript
ORF <- GRanges("1", c(3), "+") # start site
names(ORF) <- "tx1_1" # ORF 1 on tx1
tx <- GRangesList(tx1 = GRanges("1", c(1,3,5,7,9,11,13), "+"))
windowPerGroup(ORF, tx, upstream = -3, downstream = 5) # <- 2nd codon
```

`windowPerReadLength` *Find proportion of reads per position in window*

Description

This is like a more detailed floss score, where floss score takes fraction of reads per read length over whole window, this is defined as: Fraction of reads per read length, per position in whole window (by upstream and downstream)

Usage

```
windowPerReadLength(grl, tx = NULL, reads, pShifted = TRUE,
  upstream = if (pShifted) 5 else 20, downstream = if (pShifted) 20
  else 5, acceptedLengths = NULL, zeroPosition = upstream,
  scoring = "transcriptNormalized")
```

Arguments

`grl`	a `GRangesList` object with usually either leaders, cds', 3' utrs or ORFs
`tx`	default NULL, a GRangesList of transcripts or (container region), names of tx must contain all grl names. The names of grl can also be the ORFik orf names. that is "txName_id"
`reads`	any type of reads, usualy ribo seq. As GAlignment, GRanges or GRangesList object.
`pShifted`	a logical (TRUE), are riboseq reads p-shifted to size 1 width reads? If upstream and downstream is set, this argument is irrelevant.
`upstream`	an integer (5), relative region to get upstream from.
`downstream`	an integer (20), relative region to get downstream from
`acceptedLengths`	an integer vector (NULL), the read lengths accepted. Default NULL, means all lengths accepted.
`zeroPosition`	an integer DEFAULT (upstream), the point if all windows are equal size, that should be set to position 0. Like leaders and cds combination, then 0 is the TIS and -1 is last base in leader. NOTE!: if windows have different widths, this will be ignored.
`scoring`	a character (transcriptNormalized), one of (zscore, transcriptNormalized, mean, median, sum, sumLength, fracPos), see ?coverageScorings. Use to choose meta coverage or per transcript.

Details

If tx is not NULL, it gives a metaWindow, centered around startSite of grl from upstream and downstream. If tx is NULL, it will use only downstream , since it has no reference from to find upstream from. Unless upstream is negative, that is, going downstream.

Value

a data.frame with lengths by coverage / vector of proportions

See Also

Other coverage: coverageScorings, metaWindow, scaledWindowPositions

---

`windowPerTranscript` *Get coverage window per transcript*

Description

NOTE: All ranges with smaller width than windowSize, will of course be removed. What is the 100 position on a 1 width size ?

Usage

```
windowPerTranscript(txdb, reads, splitIn3 = TRUE, windowSize = 100,
  fraction = "1")
```

Arguments

`txdb`	a TxDb object or a path to gtf/gff/db file.
`reads`	GRanges or GAlignment of reads
`splitIn3`	a logical(TRUE), split window in 3 (leader, cds, trailer)
`windowSize`	an integer (100), size of windows
`fraction`	a character (1), info on reads (which read length, or which type (RNA seq))

Value

a data.table with columns position, score

`xAxisScaler` *Scale x axis correctly*

Description

Works for all coverage plots, that need 0 position aligning

Usage

```
xAxisScaler(covPos)
```

Arguments

`covPos`	a numeric vector of positions in coverage

Value

a numeric vector from the seq() function, aligned to 0.

`yAxisScaler` *Scale y axis correctly*

Description

Works for all coverage plots.

Usage

```
yAxisScaler(covPos)
```

Arguments

`covPos`	a levels object from a factor of y axis

Value

a character vector from the seq() function, aligned to 0.

Index