Package 'ORFik'

February 19, 2019

Type Package

Title Open Reading Frames in Genomics

Version 1.2.1

Encoding UTF-8

Description Tools for manipulation of 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 the whole genomes and many more.

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

License MIT + file LICENSE

LazyData TRUE

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

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

Depends R (>= 3.5.0), IRanges (>= 2.13.28), GenomicRanges (>= 1.31.23), GenomicAlignments (>= 1.15.13)

Imports S4Vectors (>= 0.17.39), GenomeInfoDb (>= 1.15.5), GenomicFeatures (>= 1.31.10), rtracklayer (>= 1.39.9), Rcpp (>= 0.12.16), data.table (>= 1.10.4-3), Biostrings (>= 2.47.12), stats, tools, Rsamtools (>= 1.31.3), BiocGenerics (>= 0.25.3)

RoxygenNote 6.1.0

Suggests testthat, rmarkdown, knitr, BiocStyle, BSgenome, BSgenome.Hsapiens.UCSC.hg19, ggplot2 (>= 2.2.1)

LinkingTo Rcpp

VignetteBuilder knitr

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

git_branch RELEASE_3_8

git_last_commit 2e3a8c6

git_last_commit_date 2019-01-04

Date/Publication 2019-02-18

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

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

Authors:

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

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/JokingHero/ORFik
- Report bugs at https://github.com/JokingHero/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

`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)
```

Arguments

`fiveUTRs`	The 5' leader sequences as GRangesList
`maxPeakPosition`	The max peak for each 5' leader found by cage

Value

a GRanges object of first exons

`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 splits, you can use exon prediction tools, or run sequencing experiments.

Usage

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

Arguments

txdb	a TxDb object, normally from a gtf file.
cage	Either a filePath for CageSeq file, or already loaded CageSeq peak data as GRanges.
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 you want to restrict leaders to not extend closer than 5 bases from closest upstream leader, set this to TRUE.

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")
txdb <- loadDb(txdbFile)
cagePath <- system.file("extdata", "cage-seq-heart.bed.bgz",
package = "ORFik")
reassignTxDbByCage(txdb, cagePath)

## End(Not run)
```

---

asTX *Map genomic to transcript coordinates by reference*

Description

Map genomic to transcript coordinates by reference

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, pmapFromTranscriptF, reduceKeepAttr, tile1, txSeqsFromFa, windowPerGroup

---

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, is.gr_or_grl, is.grl, validGRL

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 points to analyse, assumes the zero is in the middle + 1
feature	(character) either "start" or "stop"

Value

a single numeric offset

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

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

codonSumsPerGroup *Get 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 Example: c(1,0,0,1), with reg_len = 2, gives c(1,0) and c(0,1), these are summed and returned as list

Usage

```
codonSumsPerGroup(countList, reg_len, runLengths)
```

Arguments

countList	a Rle of count repetitions (000,1,00,1 etc)
reg_len	integer vector, size of runs
runLengths	integer vector, duplications per run

Value

a list of 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 = 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,
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. In the future release reasignment will create txdb objects for you, but currently this is not supported, therefore be carefull.

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, subsetCoverage, translationalEff

Examples

```
# Usually the ORFs are found in orfik, which makes names for you etc.
# Here we make an example from scratch
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 correct lists, 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,
`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. If you do have a txdb with e.g. cage reassignments, use computeFeatures instead. 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, 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 3 ways of doing this 1. Take 5' ends, reduce away rest (5prime) 2. Tile and include all (tileAll) 3. Take middle point per GRanges (middle)

Usage

```
convertToOneBasedRanges(gr, method = "5prime", addScoreColumn = FALSE)
```

Arguments

`gr`	GRanges 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 position.

Value

Converted GRanges object

See Also

Other utils: `bedToGR`, `findFa`, `fread.bed`, `is.gr_or_grl`, `is.grl`, `validGRL`

coverageByWindow *Compute coverage for every GRangesList subset.*

Description

This is similar to [GenomicFeatures::coverageByTranscript()], but it adds: automatic sorting of the windows, fix for some rare cases when subsetting fails on minus/plus strands and security that subseting of windows will always return values (zeros) istead of out of bounds error.

Usage

```
coverageByWindow(x, windows, ignore.strand = FALSE, is.sorted = FALSE,
  keep.names = TRUE)
```

Arguments

x	the cigar of the reads
windows	(GRangesList) of transcripts or CDS or other ranges that will be subseting coverage of ‘x‘
ignore.strand	(logical) Whether to consider all reads to be "*".
is.sorted	(logical), is windows already sorted.
keep.names	logical (T), keep names and meta cols

Details

Minus strand is already flipped so that the most 5’ position on the window is the first position in the returned Rle.

Value

(RleList) of positional counts of ‘x‘ ranges overlapping each consecutive position of the elements of ‘windows‘

Examples

```
cds <- GenomicRanges::GRangesList(
  GenomicRanges::GRanges(seqnames = "chr1",
                         ranges = IRanges::IRanges(100, 200),
                         strand = "+"))
reads <- GenomicRanges::GRanges(
  seqnames = "chr1",
  ranges =  IRanges::IRanges(c(100, 150), c(110, 160)),
  strand = "+")
ORFik:::coverageByWindow(reads, cds)
```

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)
```

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.

Value

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

See Also

Other ExtendGenomicRanges: asTX, overlapsToCoverage, pmapFromTranscriptF, 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)
```

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)
```

Arguments

footprints	(GAlignments) object of RiboSeq reads - footprints
txdb	a txdb object from a gtf file
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.
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
minCDS	(integer) minimum bp for CDS during filtering for the transcripts
minThreeUTR	(integer) minimum bp for 3' UTR during filtering for the transcripts
firstN	(integer) Represents how many bases of the transcripts downstream of start codons to use for initial estimation of the periodicity.

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.

Value

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

Examples

```
## Not run:
gtf_file <- system.file("extdata", "annotations.gtf", package = "ORFik")
txdb <- GenomicFeatures::makeTxDbFromGFF(gtf_file, format = "gtf")
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, txdb, stop = TRUE)

## End(Not run)
```

---

disengagementScore *Disengagement score (DS)*

Description

Disengagement score is defined as

```
(RPFs over ORF)/(RPFs downstream to tx 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), have you ran this line: RFP <- sort(RFP[countOverlaps(RFP, tx, type = "within" 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, 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. Matching is done by transcript names. This is applicable practically to the upstream (fiveUTRs) ORFs. 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, 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 the upstream (fiveUTRs) ORFs. If ORF is not within specified search space in fiveUTRs, this function will crash.

Usage

```
distToTSS(ORFs, fiveUTRs)
```

Arguments

ORFs	orfs as GRangesList, names of orfs must be txname_[rank]
fiveUTRs	5' leaders 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, subsetCoverage, translationalEff

Examples

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

`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

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.*

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 object or GRanges or GRangesList, usualy data from RiboSeq or RnaSeq

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, subsetCoverage, translationalEff

Examples

```
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)
RFP <- 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, RFP)
# or on cds
cdsORF <- GRanges("1", IRanges(35, 44), "+", names = "tx1")
names(cdsORF) <- "tx1"
cds <-  GRangesList(tx1 = cdsORF)
entropy(cds, RFP)
```

---

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 = 1000, cds = NULL)
```

Arguments

grl	a GRangesList of 5’ utrs or transcripts.
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

---

filterCage *Filter peak of cage-data by value*

Description

Filter peak of cage-data by value

Usage

```
filterCage(rawCage, filterValue = 1, fiveUTRs = NULL)
```

Arguments

rawCage	The raw cage-data
filterValue	The number of counts(score) to filter on for a tss to pass as hit
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.

Value

the filtered Granges object

---

findCageUTRFivelen *Get length of leaders ordered after oldTxNames*

Description

Normally only a helper function for ORFik

Usage

```
findCageUTRFivelen(fiveUTRs, oldTxNames)
```

Arguments

fiveUTRs	a GRangesList object of leaders
oldTxNames	a character vector of names to group fiveUTRs by.

Value

a GRangesList of reordered leaders.

---

findFa *Convenience wrapper for Rsamtools FaFile*

Description

Convenience wrapper for Rsamtools FaFile

Usage

```
findFa(faFile)
```

Arguments

faFile	a character path or FaFile

Value

a FaFile or BSgenome

See Also

Other utils: bedToGR, convertToOneBasedRanges, fread.bed, is.gr_or_grl, is.grl, validGRL

`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, [findMapORFs()] 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) DNA 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 regions.
`startCodon`	(character) Possible START codons to search for. Check startDefinition for helper function.
`stopCodon`	(character) 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. 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.

Value

A GRangesList of ORFs.

See Also

[findORFs()], [findORFsFasta()], [startDefinition()], [stopDefinition()]

Other findORFs: `findORFsFasta`, `findORFs`, `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) DNA 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 regions.
startCodon	(character) Possible START codons to search for. Check startDefinition for helper function.

stopCodon	(character) 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. 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 (DNAStrin neg <- findORFs(reverseComplement(DNAStringSet(seqs))) relist(c(GRanges(pos, strand = "+"), GRang`

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

[findMapORFs()], [findORFsFasta()], [startDefinition()], [stopDefinition()]

Other findORFs: findMapORFs, findORFsFasta, 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 eukaryotes, 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) Possible START codons to search for. Check startDefinition for helper function.
stopCodon	(character) 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. 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

[findORFs()], [findMapORFs()], [startDefinition()], [stopDefinition()]

Other findORFs: findMapORFs, findORFs, startDefinition, stopDefinition

Examples

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

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, 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". When calculating RiboSeq data FPKM over ORFs use ORFs as 'grl'. When calculating RNASeq data FPKM use full transcripts as 'grl'.

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).
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.

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, 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 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`, `subsetCoverage`, `translationalEff`

`fractionLength` *Fraction Length*

Description

Fraction Length is defined as

`(lengths of grl)/(length of 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 graction 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`, `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.

Usage

```
fread.bed(filePath)
```

Arguments

filePath	The location of the bed file

Value

a GRanges object

See Also

Other utils: bedToGR, convertToOneBasedRanges, findFa, is.gr_or_grl, is.grl, validGRL

Examples

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

getStartStopWindows *Get Start and Stop codon within specified windows over CDS.*

Description

For each cds in ‘txdb‘ object, filtered by ‘txNames‘, get a window around start and stop codons within ‘window_size‘ downstream and upstream of the codon.

Usage

```
getStartStopWindows(txdb, txNames, start = TRUE, stop = TRUE,
  window_size = 30L, cds = NULL)
```

Arguments

txdb	a txdb object of annotations
txNames	a character vector of the transcript names to use
start	(logical) whether to include start codons
stop	(logical) whether to incude stop codons
window_size	(integer) size of the window to extract upstream of the start/stop codon and downstream
cds	a GRangesList with cds, a speedup if you already have them loaded.

Value

a list with two slots "starts" and "stops", each contains a GRangesList of windows around start and stop codons for the transcripts of interest

Examples

```
gtf_file <- system.file("extdata", "annotations.gtf", package = "ORFik")
txdb <- GenomicFeatures::makeTxDbFromGFF(gtf_file, format = "gtf")
txNames <- txNamesWithLeaders(txdb)
getStartStopWindows(txdb, txNames)
```

---

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

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
```

---

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

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, footprints, pShifted = TRUE)
```

Arguments

grl	a GRangesList object with ORFs
cds	a GRangesList object with coding sequences
tx	a GrangesList of transcripts covering grl.
footprints	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

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, subsetCoverage, translationalEff

Examples

```
# Good hiting ORF
ORF <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(21), end = c(40)),
               strand = "+")
names(ORF) <- c("tx1")
grl <- GRangesList(tx1 = ORF)
# 1 width position based
RFP <- GRanges("1", IRanges(c(21, 23, 50, 50, 50, 53, 53, 56, 59),
 c(21, 23, 50, 50, 50, 53, 53, 56, 59)), "+")
score(RFP) <- 28 # original width
cds <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(50), end = c(80)),
               strand = "+")
cds <- GRangesList(tx1 = cds)
tx <- GRanges(seqnames = "1",
               ranges = IRanges(1,85),
               strand = "+")
tx <- GRangesList(tx1 = tx)

initiationScore(grl, cds, tx, RFP, 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 was 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") 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, 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 utils: bedToGR, convertToOneBasedRanges, findFa, fread.bed, is.gr_or_grl, validGRL

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 utils: bedToGR, convertToOneBasedRanges, findFa, fread.bed, is.grl, validGRL

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

Description

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

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`, `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, 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

Find if there is periodicity in the vector

Usage

```
isPeriodic(x)
```

Arguments

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

Value

a logical, if it is periodic.

---

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, faFile, species = "human", include.N = FALSE)
```

Arguments

grl	a GRangesList grouped by ORF
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.

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, 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
# get faFile for sequences
faFile <- FaFile(system.file("extdata", "genome.fasta", package = "ORFik"))
kozakSequenceScore(ORFs, 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)
```

`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 a meta column with exon ranks*

Description

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

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

`metaWindow` *Calculate metaplot coverage of reads around input GRangesList object.*

Description

Sums up coverage over set of GRanges objects that.

Usage

```
metaWindow(x, windows)
```

Arguments

`x`	GRanges object of your reads. You should resize them beforehand to width of 1 to focus on 5' ends of footprints.
`windows`	GRanges object of your CDSs start or stop postions. Its width has to be even number as we will assume in the middle is position zero which is included in the downstream window.

Value

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

Examples

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

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)
```

orfID *Get id's for 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	ribozomal 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.

Value

a matrix with 4 columns, the orfscore and score of each of the 3 tiles

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, 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)
```

---

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 of 5' utrs or transcripts.
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, pmapFromTranscriptF, 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 numeric
is_plus_strand	logical

Value

the shifted read

pmapFromTranscriptF *Faster more secure version of mapFromTranscripts*

Description

Fixes a bug in function, and should have 10x speedup Also removes hit column for you

Usage

```
pmapFromTranscriptF(ranges, grl, indices)
```

Arguments

ranges	IRanges of ranges within grl
grl	the "transcripts" that contain ranges, GRangesList
indices	integer vector of which index of grl ranges are from: (c(1,1,2)) means first two ranges are from grl[1], third from grl[2])

Value

A GrangesList of ranges mapped from transcripts

See Also

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

rankOrder *ORF rank in transcripts*

Description

ig. second orf _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, 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)
```

readWidths *Get RiboSeq widths*

Description

Input a 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)
```

Arguments

reads	a GRanges or GAlignment object.

Details

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

Value

an integer vector of widths

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).

Usage

```
reassignTSSbyCage(fiveUTRs, cage, extension = 1000, filterValue = 1,
  restrictUpstreamToTx = FALSE)
```

Arguments

fiveUTRs	(GRangesList) The 5' leaders or transcript sequences
cage	Either a filePath for CageSeq file, or already loaded CageSeq peak data as GRanges.
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 you want to restrict leaders to not extend closer than 5 bases from closest upstream leader, set this to TRUE.

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 will be fixed by ORFik
# finally reassign TSS for fiveUTRs
reassignTSSbyCage(fiveUTRs, cage)
```

---

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)
```

Arguments

txdb	a TxDb object, normally from a gtf file.
cage	Either a filePath for CageSeq file, or already loaded CageSeq peak data as GRanges.
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 you want to restrict leaders to not extend closer than 5 bases from closest upstream leader, set this to TRUE.

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")
txdb <- loadDb(txdbFile)
cagePath <- system.file("extdata", "cage-seq-heart.bed.bgz",
package = "ORFik")
reassignTxDbByCage(txdb, 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 loose 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, pmapFromTranscriptF, 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)
```

---

regroupRleList *Regroup rle from GRangesList*

Description

Almost direct copy of IRanges regroupBySupergroup. But only works on rle and GRangesList. This function will be removed if IRanges regroupBySupergroup is exported.

Usage

```
regroupRleList(rle, supergroups)
```

Arguments

rle	A RleList to reduce groups on.
supergroups	A GRangesList to group by

Value

A regrouped RleList

---

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

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)

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.

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`, `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`, `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)
```

`riboTISCoverageProportion` *Find proportion of reads per position in ORF TIS window*

Description

Proportion defined as: average count per position in -20,20 normalized by counts per gene

Usage

```
riboTISCoverageProportion(grl, tx, footprints, onlyProportion = FALSE,
  average = FALSE, pShifted = TRUE, keep.names = FALSE,
  upStart = if (pShifted) 5 else 20, downStop = if (pShifted) 20 else
  5)
```

Arguments

`grl`	a `GRangesList` object with usually new ORFs, but can also be either leaders, cds', 3'UTRs.
`tx`	a GrangesList of transcripts covering grl.
`footprints`	ribo seq reads as GAlignment, GRanges or GRangesList object.
`onlyProportion`	a logical (FALSE), return whole data.frame or only proportions
`average`	a logical (FALSE), take average over coverage in all grl ?
`pShifted`	a logical (TRUE), are riboseq reads p-shifted?
`keep.names`	a logical(FALSE), only applies when onlyProportion is TRUE.
`upStart`	upstream region boundary (5 or 20 as standard), relative (5, mean 5 upstream from TIS)
`downStop`	downstream region boundary (5 or 20 as standard), relative (5, mean 5 downstream from TIS)

Details

This pattern can be averaged on CDS's, to find other ORFs. When detecting new ORFs, this CDS average can be used as a template.

Value

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

`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, selected_lengths, selected_shifts)
```

Arguments

footprints	(GAlignments) object of RiboSeq reads
selected_lengths	Numeric vector of lengths of footprints you select for shifting.
selected_shifts	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.

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.

Examples

```
## Not run:
gtf_file <- system.file("extdata", "annotations.gtf", package = "ORFik")
txdb <- GenomicFeatures::makeTxDbFromGFF(gtf_file, format = "gtf")
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, txdb)
# shift the RiboSeq footprints
shiftedReads <- shiftFootprints(footprints, shifts$fragment_length,
                                shifts$offsets_start)

## End(Not run)
```

---

sortPerGroup *Sort a GRangesList*

Description

A faster, more versatile reimplementation of sort.GenomicRanges for GRangesList, which 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)
```

---

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

Description

In ATGTTTTGC, 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 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`, `stopDefinition`

Examples

```
startDefinition
startDefinition(1)
```

startRegionString *Get -20,20 start region as DNA characters per gr group*

Description

Get -20,20 start region as DNA characters per gr group

Usage

```
startRegionString(grl, tx, faFile, groupBy = NULL)
```

Arguments

grl	a GRangesList to find regions
tx	a GRangesList of transcripts containing grl
faFile	a FaFile from the fasta file, see ?FaFile. Can also be path to fastaFile with fai file in same dir.
groupBy	(NULL) column to group grl by, if NULL group by names(gr)

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	if asGR is False, do you still want to keep a named vector
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 ATGTTTTGC, get the positions TGC. 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 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, 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	if asGR is False, do you still want to keep a named vector
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, translationalEff

---

`subset_to_frame` *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

```
subset_to_frame(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 a GRangesList by 1*

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. This is not supported originally by GenomicRanges. As a precaution, this function requires the unlisted objects to have names.

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, pmapFromTranscriptF, 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, 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
```

---

txLen *Get transcript lengths*

Description

A helper function for easy length retrieval

Usage

```
txLen(Gtf = NULL, changedFiveUTRs = NULL)
```

Arguments

Gtf	a TxDb object of a gtf file
changedFiveUTRs	a GRangesList object of leaders. Only add this if you used cage data or other things to change the leaders, therefor we need it to update transcript lengths.

Value

a vector of transcript lengths

txNames *Get transcript names from orf names*

Description

names must either be a column called names, or the names of the grl object

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

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)
```

txNamesWithLeaders *Get the transcripts that have minimum lengths of leaders and cds.*

Description

Filter transcripts to those whohave 5' UTR, CDS, 3' UTR of some lengths, pick the longest per gene.

Usage

```
txNamesWithLeaders(txdb, minFiveUTR = 30L, minCDS = 150L,
  minThreeUTR = 30L)
```

Arguments

txdb	a TxDb object from gtf
minFiveUTR	(integer) minimum bp for 5' UTR during filtering for the transcripts
minCDS	(integer) minimum bp for CDS during filtering for the transcripts
minThreeUTR	(integer) minimum bp for 3' UTR during filtering for the transcripts

Value

a character vector of valid tramscript names

Examples

```
gtf_file <- system.file("extdata", "annotations.gtf", package = "ORFik")
txdb <- GenomicFeatures::makeTxDbFromGFF(gtf_file, format = "gtf")
txNames <- txNamesWithLeaders(txdb)
```

---

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

Description

A small safety wrapper around extractTranscriptSeqs

Usage

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

Arguments

grl	a GRangesList object
faFile	FaFile or BSgenome used to find the transcripts,
is.sorted	a speedup, if you know the ranges are sorted

Value

a DNAStringSet of the transcript sequences

See Also

Other ExtendGenomicRanges: asTX, coveragePerTiling, overlapsToCoverage, pmapFromTranscriptF, 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. One difference is that if grl have no names, it will not make integer names, but keep them as null.

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. 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, extension = 1000, filterValue = 1,
  cds = NULL)
```

Arguments

fiveUTRs	(GRangesList) The 5' leaders or transcript sequences
cage	Either a filePath for CageSeq file, or already loaded CageSeq peak data as GRanges.
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.
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.

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)
```

Arguments

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

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

---

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 utils: bedToGR, convertToOneBasedRanges, findFa, fread.bed, is.gr_or_grl, is.grl

---

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)
```

---

windowPerGroup *Get window region of tx around point of gr*

Description

If downstreamFrom is 20, it means the window will start -20 downstream of gr start site. It will keep exon structure of tx

Usage

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

Arguments

gr	a GRanges object (startSites and others, must be single point)
tx	a GRangesList of transcripts or (container region)
downstream	an integer, relative region to get downstream from
upstream	an integer vector, relative region to get upstream from.

Value

a GRanges/GRangesList object if exon/introns

See Also

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

Index