# Package: ORFik (via r-universe)

June 21, 2024

**Type** Package

**Title** Open Reading Frames in Genomics

**Version** 1.25.3

**Encoding** UTF-8

**Description** R package for analysis of transcript and translation features through manipulation of sequence data and NGS data like Ribo-Seq, RNA-Seq, TCP-Seq and CAGE. It is generalized in the sense that any transcript region can be analysed, as the name hints to it was made with investigation of ribosomal patterns over Open Reading Frames (ORFs) as it's primary use case. ORFik is extremely fast through use of C++, data.table and GenomicRanges. Package allows to reassign starts of the transcripts with the use of CAGE-Seq data, automatic shifting of RiboSeq reads, finding of Open Reading Frames for whole genomes and much more.

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

**License** MIT + file LICENSE

**LazyData** TRUE

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

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

**Depends** R (>= 4.1.0), IRanges (>= 2.17.1), GenomicRanges (>= 1.35.1), GenomicAlignments (>= 1.19.0)

**Imports** AnnotationDbi (>= 1.45.0), Biostrings (>= 2.51.1), biomaRt, biomartr (>= 1.0.7), BiocFileCache, BiocGenerics (>= 0.29.1), BiocParallel (>= 1.19.0), BSgenome, cowplot (>= 1.0.0), data.table (>= 1.11.8), DESeq2 (>= 1.24.0), fst (>= 0.9.2), GenomeInfoDb (>= 1.15.5), GenomicFeatures (>= 1.31.10), ggplot2 (>= 2.2.1), gridExtra (>= 2.3), httr (>= 1.3.0), jsonlite, methods (>= 3.6.0), R.utils, Rcpp (>= 1.0.0), Rsamtools (>= 1.35.0), rtracklayer (>= 1.43.0), stats, SummarizedExperiment (>= 1.14.0), S4Vectors (>= 0.21.3), txdbmaker, tools, utils, XML, xml2 (>= 1.2.0), withr

**RoxygenNote** 7.3.1

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

**LinkingTo** Rcpp

**VignetteBuilder** knitr

**Repository** https://bioc.r-universe.dev

**RemoteUrl** https://github.com/bioc/ORFik

**RemoteRef** HEAD

**RemoteSha** 7434daf26cbcaa1bc683af1997dca57209eb6404

# Contents

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

## Description

Main goals:

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

## Author(s)

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

Authors:

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

Other contributors:

- Michal Swirski <michal.swirski@uw.edu.pl> [contributor]
- Katarzyna Chyzynska <katchyz@gmail.com> [contributor, data contributor]
- Yamila Torres Cleuren <yamilatorrescleuren@gmail.com> [contributor, thesis advisor]
- Eivind Valen <eivind.valen@gmail.com> [thesis advisor, funder]

## See Also

Useful links:

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

`artificial.orfs` *Create small artificial orfs from cds*

## Description

Usefull to see if short ORFs prediction is dependent on length.
Split cds first in two, a start part and stop part. Then say how large the two parts can be and merge them together. It will sample a value in range give.
Parts will be forced to not overlap and can not extend outside original cds

## Usage

```
artificial.orfs(
  cds,
  start5 = 1,
  end5 = 4,
  start3 = -4,
  end3 = 0,
  bin.if.few = TRUE
)
```

## Arguments

| | |
|---|---|
| `cds` | a GRangesList of orfs, must have width %% 3 == 0 and length >= 6 |
| `start5` | integer, default: 1 (start of orf) |
| `end5` | integer, default: 4 (max 4 codons from start codon) |
| `start3` | integer, default -4 (max 4 codons from stop codon) |
| `end3` | integer, default: 0 (end of orf) |
| `bin.if.few` | logical, default TRUE, instead of per codon, do per 2, 3, 4 codons if you have few samples compared to lengths wanted, If you have 4 cds' and you want 7 different lengths, which is the standard, it will give you possible nt length: 6-12-18-24 instead of original 6-9-12-15-18-21-24.<br>If you have more than 30x cds than lengths wanted this is skipped. (for default arguments this is: 7*30 = 210 cds) |

## Details

If artificial cds length is not divisible by 2, like 3 codons, the second codon will always be from the start region etc.
Also If there are many very short original cds, the distribution will be skewed towards more smaller artificial cds.

## Value

GRangesList of new ORFs (sorted: + strand increasing start, - strand decreasing start)

## Examples

```
txdb <- ORFik.template.experiment()
#cds <- loadRegion(txdb, "cds")
## To get enough CDSs, just replicate them
# cds <- rep(cds, 100)
#artificial.orfs(cds)
```

---

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

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| `txdb` | a TxDb file, a path to one of: (.gtf ,.gff, .gff2, .gff2, .db or .sqlite) or an ORFik experiment |
| `cage` | Either a filePath for the CageSeq file as .bed .bam or .wig, with possible compressions (".gzip", ".gz", ".bgz"), or already loaded CageSeq peak data as GRanges or GAlignment. NOTE: If it is a .bam file, it will add a score column by running: convertToOneBasedRanges(cage, method = "5prime", addScoreColumn = TRUE) The score column is then number of replicates of read, if score column is something else, like read length, set the score column to NULL first. |
| `extension` | The maximum number of basses upstream of the TSS to search for CageSeq peak. |
| `filterValue` | The minimum number of reads on cage position, for it to be counted as possible new tss. (represented in score column in CageSeq data) If you already filtered, set it to 0. |

`restrictUpstreamToTx`
: a logical (FALSE). If TRUE: restrict leaders to not extend closer than 5 bases from closest upstream leader, set this to TRUE.

`removeUnused`
: logical (FALSE), if False: (standard is to set them to original annotation), If TRUE: remove leaders that did not have any cage support.

`preCleanup`
: logical (TRUE), if TRUE, remove all reads in region (-5:-1, 1:5) of all original tss in leaders. This is to keep original TSS if it is only +/- 5 bases from the original.

`pseudoLength`
: a numeric, default 1. Either if no CAGE supports the leader, or if CAGE is set to NULL, add a pseudo length for all the UTRs. Will not extend a leader if it would make it go outside the defined seqlengths of the genome. So this length is not guaranteed for all!

## 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). If no CAGE supports a leader, the width will be set to 1 base.

## Value

a TxDb obect of reassigned transcripts

## See Also

Other CAGE: `reassignTSSbyCage()`, `reassignTxDbByCage()`

## Examples

```
txdbFile <- system.file("extdata", "hg19_knownGene_sample.sqlite",
 package = "GenomicFeatures")
cagePath <- system.file("extdata", "cage-seq-heart.bed.bgz",
 package = "ORFik")

## Not run:
  assignTSSByCage(txdbFile, cagePath)
  #Minimum 20 cage tags for new TSS
  assignTSSByCage(txdbFile, cagePath, filterValue = 20)
  # Create pseudo leaders for the ones without hits
  assignTSSByCage(txdbFile, cagePath, pseudoLength = 100)
  # Create only pseudo leaders (in example 2 leaders are added)
  assignTSSByCage(txdbFile, cage = NULL, pseudoLength = 100)

## End(Not run)
```

asTX *Map genomic to transcript coordinates by reference*

## Description

Map range coordinates between features in the genome and transcriptome (reference) space.

## Usage

```
asTX(
  grl,
  reference,
  ignore.strand = FALSE,
  x.is.sorted = TRUE,
  tx.is.sorted = TRUE
)
```

## 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 |
| ignore.strand | When ignore.strand is TRUE, strand is ignored in overlaps operations (i.e., all strands are considered "+") and the strand in the output is '*'.<br>When ignore.strand is FALSE (default) strand in the output is taken from the transcripts argument. When transcripts is a GRangesList, all inner list elements of a common list element must have the same strand or an error is thrown.<br>Mapped position is computed by counting from the transcription start site (TSS) and is not affected by the value of ignore.strand. |
| x.is.sorted | if x is a GRangesList object, are "-" strand groups pre-sorted in decreasing order within group, default: TRUE |
| tx.is.sorted | if transcripts is a GRangesList object, are "-" strand groups pre-sorted in decreasing order within group, default: TRUE |

## Details

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

## Value

a GRangesList in transcript coordinates

## See Also

Other ExtendGenomicRanges: coveragePerTiling(), extendLeaders(), extendTrailers(), reduceKeepAttr(), tile1(), txSeqsFromFa(), windowPerGroup()

## Examples

```
seqname <- c("tx1", "tx2", "tx3")
seqs <- c("ATGGGTATTTATA", "AAAAA", "ATGGGTAATA")
grIn1 <- GRanges(seqnames = "1",
                 ranges = IRanges(start = c(21, 10), end = c(23, 19)),
                 strand = "-")
grIn2 <- GRanges(seqnames = "1",
                 ranges = IRanges(start = c(1), end = c(5)),
                 strand = "-")
grIn3 <- GRanges(seqnames = "1",
                 ranges = IRanges(start = c(1010), end = c(1019)),
                 strand = "-")
grl <- GRangesList(grIn1, grIn2, grIn3)
names(grl) <- seqname
# Find ORFs
test_ranges <- findMapORFs(grl, seqs,
                 "ATG|TGG|GGG",
                 "TAA|AAT|ATA",
                 longestORF = FALSE,
                 minimumLength = 0)
# Genomic coordinates ORFs
test_ranges
# Transcript coordinate ORFs
asTX(test_ranges, reference = grl)
# seqnames will here be index of transcript it came from
```

---

# bamVarName *Get library variable names from ORFik* experiment

---

## Description

What will each sample be called given the columns of the experiment? A column is included if more than 1 unique element value exist in that column.

## Usage

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

```
  skip.libtype = FALSE,
  fraction_prepend_f = TRUE
)
```

## Arguments

| | |
|---|---|
| df | an ORFik experiment |
| skip.replicate | a logical (FALSE), don't include replicate in variable name. |
| skip.condition | a logical (FALSE), don't include condition in variable name. |
| skip.stage | a logical (FALSE), don't include stage in variable name. |
| skip.fraction | a logical (FALSE), don't include fraction |
| skip.experiment | a logical (FALSE), don't include experiment |
| skip.libtype | a logical (FALSE), don't include libtype |
| fraction_prepend_f | a logical (TRUE), include "f" in front of fraction, useful for knowing what fraction is. |

## Value

variable names of libraries (character vector)

## See Also

Other ORFik_experiment: ORFik.template.experiment(), ORFik.template.experiment.zf(), create.experiment(), experiment-class, filepath(), libraryTypes(), organism, experiment-method, outputLibs(), read.experiment(), save.experiment(), validateExperiments()

## Examples

```
df <- ORFik.template.experiment()
bamVarName(df)

## without libtype
bamVarName(df, skip.libtype = TRUE)
## Without experiment name
bamVarName(df, skip.experiment = TRUE)
```

---

# browseSRA *Open SRA in browser for specific bioproject*

---

## Description

Open SRA in browser for specific bioproject

### Usage

```
browseSRA(x, browser = getOption("browser"))
```

### Arguments

| | |
|---|---|
| x | character, bioproject ID. |
| browser | a non-empty character string giving the name of the program to be used as the HTML browser. It should be in the PATH, or a full path specified. Alternatively, an R function to be called to invoke the browser.<br>Under Windows NULL is also allowed (and is the default), and implies that the file association mechanism will be used. |

### Value

invisible(NULL), opens webpage only

### See Also

Other sra: download.SRA(), download.SRA.metadata(), download.ebi(), get_bioproject_candidates(), install.sratoolkit(), rename.SRA.files()

### Examples

```
#browseSRA("PRJNA336542")

#' # For windows make sure a valid browser is defined:
browser <- getOption("browser")
#browseSRA("PRJNA336542", browser)
```

---

## codon_usage *Codon usage*

---

### Description

Per AA / codon, analyse the coverage, get a multitude of features. For both A sites and P-sites (Input reads must be P-sites for now) This function takes inspiration from the codonDT paper, and among others returns the negative binomial estimates, but in addition many other features.

### Usage

```
codon_usage(
  reads,
  cds,
  mrna,
  faFile,
  filter_table,
  filter_cds_mod3 = TRUE,
```

```
  min_counts_cds_filter = max(min(quantile(filter_table, 0.5), 1000), 1000),
  with_A_sites = TRUE,
  aligned_position = "center",
  code = GENETIC_CODE
)
```

## Arguments

| | |
|---|---|
| reads | either a single library (GRanges, GAlignment, GAlignmentPairs), or a list of libraries returned from `outputLibs(df)` with p-sites. If list, the list must have names coresponding to the library names. |
| cds | a GRangesList |
| mrna | a GRangesList |
| faFile | a FaFile from genome |
| filter_table | a matrix / vector of length equal to cds |
| filter_cds_mod3 | logical, default TRUE. Remove all ORFs that are not mod3, this speeds up the computation a lot, and usually removes malformed ORFs you would not want anyway. |
| min_counts_cds_filter | numeric, default: `max(min(quantile(filter_table, 0.50), 100), 100)`. Minimum number of counts from the 'filter_table' argument. |
| with_A_sites | logical, default TRUE. Not used yet, will also return A site scores. |
| aligned_position | what positions should be taken to calculate per-codon coverage. By default: "center", meaning that positions -1,0,1 will be taken. Alternative: "left", then positions 0,1,2 are taken. |
| code | a named character vector of size 64. Default: GENETIC_CODE. Change if organism does not use the standard code. |

## Details

The primary column to use is "mean_txNorm", this is the fair normalized score.

## Value

a data.table of rows per codon / AA. All values are given per library, per site (A or P), sorted by the mean_txNorm_percentage column of the first library in the set, the columns are:

- variable (character)Library name
- seq (character)Amino acid:codon
- sum (integer)total counts per seq
- sum_txNorm (integer)total counts per seq normalized per tx
- var (numeric)variance of total counts per seq
- N (integer)total number of codons of that type

- mean_txNorm (numeric)Default use output, the fair codon usage, normalized both for gene and genome level for codon and read counts
- ...
- alpha (numeric)dirichlet alpha MOM estimator (imagine mean and variance of probability in 1 value, the lower the value, the higher the variance, mean is decided by the relative value between samples)
- sum_txNorm (integer)total counts per seq normalized per tx
- relative_to_max_score (integer)Percentage use of codon
- type (factor(character))Either "P" or "A"

## References

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7196831/

## See Also

Other codon: codon_usage_exp(), codon_usage_plot()

## Examples

```
df <- ORFik.template.experiment()[9:10,] # Subset to 2 Ribo-seq libs

## For single library
reads <- fimport(filepath(df[1,], "pshifted"))
cds <- loadRegion(df, "cds", filterTranscripts(df))
mrna <- loadRegion(df, "mrna", names(cds))
filter_table <- assay(countTable(df, type = "summarized")[names(cds)])
faFile <- findFa(df)
res <- codon_usage(reads, cds, mrna, faFile = faFile,
             filter_table = filter_table, min_counts_cds_filter = 10)
```

---

# codon_usage_exp *Codon analysis for ORFik experiment*

---

## Description

Per AA / codon, analyse the coverage, get a multitude of features. For both A sites and P-sites (Input reads must be P-sites for now) This function takes inspiration from the codonDT paper, and among others returns the negative binomial estimates, but in addition many other features.

## Usage

```
codon_usage_exp(
  df,
  reads,
  cds = loadRegion(df, "cds", filterTranscripts(df)),
  mrna = loadRegion(df, "mrna", names(cds)),
```

```
  filter_cds_mod3 = TRUE,
  filter_table = assay(countTable(df, type = "summarized")[names(cds)]),
  faFile = df@fafile,
  min_counts_cds_filter = max(min(quantile(filter_table, 0.5), 1000), 1000),
  with_A_sites = TRUE,
  code = GENETIC_CODE,
  aligned_position = "center"
)
```

## Arguments

| | |
|---|---|
| df | an ORFik experiment |
| reads | either a single library (GRanges, GAlignment, GAlignmentPairs), or a list of libraries returned from outputLibs(df) with p-sites. If list, the list must have names coresponding to the library names. |
| cds | a GRangesList, the coding sequences, default: loadRegion(df, "cds", filterTranscripts(df)), longest isoform per gene. |
| mrna | a GRangesList, the full mRNA sequences (matching by names the cds sequences), default: loadRegion(df, "mrna", names(cds)). |
| filter_cds_mod3 | logical, default TRUE. Remove all ORFs that are not mod3, this speeds up the computation a lot, and usually removes malformed ORFs you would not want anyway. |
| filter_table | an numeric(integer) matrix, where rownames are the names of the full set of mRNA transcripts. This will be subsetted to the cds subset you use. Then CDSs are filtered from this table by the 'min_counts_cds_filter' argument. |
| faFile | FaFile, BSgenome, fasta/index file path or an ORFik experiment. This file is usually used to find the transcript sequences from some GRangesList. |
| min_counts_cds_filter | numeric, default: max(min(quantile(filter_table, 0.50), 100), 100). Minimum number of counts from the 'filter_table' argument. |
| with_A_sites | logical, default TRUE. Not used yet, will also return A site scores. |
| code | a named character vector of size 64. Default: GENETIC_CODE. Change if organism does not use the standard code. |
| aligned_position | what positions should be taken to calculate per-codon coverage. By default: "center", meaning that positions -1,0,1 will be taken. Alternative: "left", then positions 0,1,2 are taken. |

## Details

The primary column to use is "mean_txNorm", this is the fair normalized score.

## Value

a data.table of rows per codon / AA. All values are given per library, per site (A or P), sorted by the mean_txNorm_percentage column of the first library in the set, the columns are:

- variable (character)Library name
- seq (character)Amino acid:codon
- sum (integer)total counts per seq
- sum_txNorm (integer)total counts per seq normalized per tx
- var (numeric)variance of total counts per seq
- N (integer)total number of codons of that type
- mean_txNorm (numeric)Default use output, the fair codon usage, normalized both for gene and genome level for codon and read counts
- ...
- alpha (numeric)dirichlet alpha MOM estimator (imagine mean and variance of probability in 1 value, the lower the value, the higher the variance, mean is decided by the relative value between samples)
- sum_txNorm (integer)total counts per seq normalized per tx
- relative_to_max_score (integer)Percentage use of codon
- type (factor(character))Either "P" or "A"

## References

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7196831/

## See Also

Other codon: `codon_usage()`, `codon_usage_plot()`

## Examples

```
df <- ORFik.template.experiment()[9:10,] # Subset to 2 Ribo-seq libs
## For single library
res <- codon_usage_exp(df, fimport(filepath(df[1,], "pshifted")),
                 min_counts_cds_filter = 10)
# mean_txNorm is adviced scoring column
# codon_usage_plot(res, res$mean_txNorm)
# Default for plot function is the percentage scaled version of mean_txNorm
# codon_usage_plot(res) # This gives check error
## For multiple libs
res2 <- codon_usage_exp(df, outputLibs(df, type = "pshifted", output.mode = "list"),
                 min_counts_cds_filter = 10)
# codon_usage_plot(res2)
```

codon_usage_plot *Plot codon_usage*

## Description

Plot codon_usage

## Usage

```
codon_usage_plot(
  res,
  score_column = res$relative_to_max_score,
  ylab = "Ribo-seq library",
  legend.position = "none",
  limit = c(0, max(score_column)),
  midpoint = limit/2,
  monospace_font = TRUE
)
```

## Arguments

| | |
|---|---|
| res | a data.table of output from a codon_usage function |
| score_column | numeric, default: res$relative_to_max_score. Which parameter to use as score column. |
| ylab | character vector, names for libraries to show on Y axis |
| legend.position | character, default "none", do not display legend. |
| limit | numeric, 2 values for plot color limits. Default: c(0, max(score_column)) |
| midpoint | numeric, default: limit/2. midpoint of color limit. |
| monospace_font | logical, default TRUE. Use monospace font, this does not work on systems (require specific font packages), set to FALSE if it crashes for you. |

## Value

a ggplot object

## See Also

Other codon: codon_usage(), codon_usage_exp()

## Examples

```
df <- ORFik.template.experiment()[9:10,] # Subset to 2 Ribo-seq libs
## For multiple libs
res2 <- codon_usage_exp(df, outputLibs(df, type = "pshifted", output.mode = "list"),
                min_counts_cds_filter = 10)
# codon_usage_plot(res2, monospace_font = TRUE) # This gives check error
codon_usage_plot(res2, monospace_font = FALSE) # monospace font looks better
```

collapse.fastq *Very fast fastq/fasta collapser*

## Description

For each unique read in the file, collapse into 1 and state in the fasta header how many reads existed of that type. This is done after trimming usually, works best for reads < 50 read length. Not so effective for 150 bp length mRNA-seq etc.

## Usage

```
collapse.fastq(
  files,
  outdir = file.path(dirname(files[1]), "collapsed"),
  header.out.format = "ribotoolkit",
  compress = FALSE,
  prefix = "collapsed_"
)
```

## Arguments

| | |
|---|---|
| files | paths to fasta / fastq files to collapse. I tries to detect format per file, if file does not have .fastq, .fastq.gz, .fq or fq.gz extensions, it will be treated as a .fasta file format. |
| outdir | outdir to save files, default: `file.path(dirname(files[1]), "collapsed")`. Inside same folder as input files, then create subfolder "collapsed", and add a prefix of "collapsed_" to the output names in that folder. |
| header.out.format | character, default "ribotoolkit", else must be "fastx". How the read header of the output fasta should be formated: ribotoolkit: ">seq1_x55", sequence 1 has 55 duplicated reads collapsed. fastx: ">1-55", sequence 1 has 55 duplicated reads collapsed |
| compress | logical, default FALSE |
| prefix | character, default "collapsed_" Prefix to name of output file. |

## Value

invisible(NULL), files saved to disc in fasta format.

## Examples

```
fastq.folder <- tempdir() # <- Your fastq files
infiles <- dir(fastq.folder, "*.fastq", full.names = TRUE)
# collapse.fastq(infiles)
```

`collapseDuplicatedReads` *Collapse duplicated reads*

## Description

For every GRanges, GAlignments read, with the same: seqname, start, (cigar) / width and strand, collapse and give a new meta column called "score", which contains the number of duplicates of that read. If score column already exists, will return input object!

## Usage

```
collapseDuplicatedReads(x, addScoreColumn = TRUE, ...)
```

## Arguments

| | |
|---|---|
| `x` | a GRanges, GAlignments or GAlignmentPairs object |
| `addScoreColumn` | logical, default: (TRUE), if FALSE, only collapse and not keep score column of counts for collapsed reads. Returns directly without collapsing if reuse.score.column is FALSE and score is already defined. |
| `...` | alternative arguments for class instances. For example, see: `?'collapseDuplicatedReads,GRanges-met` |

## Value

a GRanges, GAlignments, GAlignmentPairs or data.table object, same as input

## Examples

```
gr <- rep(GRanges("chr1", 1:10,"+"), 2)
collapseDuplicatedReads(gr)
```

`collapseDuplicatedReads,data.table-method` *Collapse duplicated reads*

## Description

For every GRanges, GAlignments read, with the same: seqname, start, (cigar) / width and strand, collapse and give a new meta column called "score", which contains the number of duplicates of that read. If score column already exists, will return input object!

### Usage

```
## S4 method for signature 'data.table'
collapseDuplicatedReads(
  x,
  addScoreColumn = TRUE,
  addSizeColumn = FALSE,
  reuse.score.column = TRUE,
  keepCigar = FALSE
)
```

### Arguments

| | |
|---|---|
| x | a GRanges, GAlignments or GAlignmentPairs object |
| addScoreColumn | logical, default: (TRUE), if FALSE, only collapse and not keep score column of counts for collapsed reads. Returns directly without collapsing if reuse.score.column is FALSE and score is already defined. |
| addSizeColumn | logical (FALSE), if TRUE, add a size column that for each read, that gives original width of read. Useful if you need original read lengths. This takes care of soft clips etc. If collapsing reads, each unique range will be grouped also by size. |
| reuse.score.column | logical (TRUE), if addScoreColumn is TRUE, and a score column exists, will sum up the scores to create a new score. If FALSE, will skip old score column and create new according to number of replicated reads after conversion. If addScoreColumn is FALSE, this argument is ignored. |
| keepCigar | logical, default FALSE. Keep the cigar information |

### Value

a GRanges, GAlignments, GAlignmentPairs or data.table object, same as input

### Examples

```
gr <- rep(GRanges("chr1", 1:10,"+"), 2)
collapseDuplicatedReads(gr)
```

---

## collapseDuplicatedReads,GAlignmentPairs-method

*Collapse duplicated reads*

---

### Description

For every GRanges, GAlignments read, with the same: seqname, start, (cigar) / width and strand, collapse and give a new meta column called "score", which contains the number of duplicates of that read. If score column already exists, will return input object!

### Usage

```
## S4 method for signature 'GAlignmentPairs'
collapseDuplicatedReads(x, addScoreColumn = TRUE)
```

### Arguments

| | |
|---|---|
| x | a GRanges, GAlignments or GAlignmentPairs object |
| addScoreColumn | logical, default: (TRUE), if FALSE, only collapse and not keep score column of counts for collapsed reads. Returns directly without collapsing if reuse.score.column is FALSE and score is already defined. |

### Value

a GRanges, GAlignments, GAlignmentPairs or data.table object, same as input

### Examples

```
gr <- rep(GRanges("chr1", 1:10,"+"), 2)
collapseDuplicatedReads(gr)
```

---

## collapseDuplicatedReads,GAlignments-method

*Collapse duplicated reads*

---

### Description

For every GRanges, GAlignments read, with the same: seqname, start, (cigar) / width and strand, collapse and give a new meta column called "score", which contains the number of duplicates of that read. If score column already exists, will return input object!

### Usage

```
## S4 method for signature 'GAlignments'
collapseDuplicatedReads(x, addScoreColumn = TRUE, reuse.score.column = TRUE)
```

### Arguments

| | |
|---|---|
| x | a GRanges, GAlignments or GAlignmentPairs object |
| addScoreColumn | logical, default: (TRUE), if FALSE, only collapse and not keep score column of counts for collapsed reads. Returns directly without collapsing if reuse.score.column is FALSE and score is already defined. |
| reuse.score.column | logical (TRUE), if addScoreColumn is TRUE, and a score column exists, will sum up the scores to create a new score. If FALSE, will skip old score column and create new according to number of replicated reads after conversion. If addScoreColumn is FALSE, this argument is ignored. |

### Value

a GRanges, GAlignments, GAlignmentPairs or data.table object, same as input

### Examples

```
gr <- rep(GRanges("chr1", 1:10,"+"), 2)
collapseDuplicatedReads(gr)
```

---

## collapseDuplicatedReads,GRanges-method

*Collapse duplicated reads*

---

### Description

For every GRanges, GAlignments read, with the same: seqname, start, (cigar) / width and strand, collapse and give a new meta column called "score", which contains the number of duplicates of that read. If score column already exists, will return input object!

### Usage

```
## S4 method for signature 'GRanges'
collapseDuplicatedReads(
  x,
  addScoreColumn = TRUE,
  addSizeColumn = FALSE,
  reuse.score.column = TRUE
)
```

### Arguments

| | |
|---|---|
| x | a GRanges, GAlignments or GAlignmentPairs object |
| addScoreColumn | logical, default: (TRUE), if FALSE, only collapse and not keep score column of counts for collapsed reads. Returns directly without collapsing if reuse.score.column is FALSE and score is already defined. |
| addSizeColumn | logical (FALSE), if TRUE, add a size column that for each read, that gives original width of read. Useful if you need original read lengths. This takes care of soft clips etc. If collapsing reads, each unique range will be grouped also by size. |
| reuse.score.column | logical (TRUE), if addScoreColumn is TRUE, and a score column exists, will sum up the scores to create a new score. If FALSE, will skip old score column and create new according to number of replicated reads after conversion. If addScoreColumn is FALSE, this argument is ignored. |

### Value

a GRanges, GAlignments, GAlignmentPairs or data.table object, same as input

### Examples

```
gr <- rep(GRanges("chr1", 1:10,"+"), 2)
collapseDuplicatedReads(gr)
```

---

`combn.pairs` *Create all unique combinations pairs possible*

---

### Description

Given a character vector, get all unique combinations of 2.

### Usage

```
combn.pairs(x)
```

### Arguments

| | |
|---|---|
| x | a character vector, will unique elements for you. |

### Value

a list of character vector pairs

### Examples

```
df <- ORFik.template.experiment()
ORFik:::combn.pairs(df[, "libtype"])
```

---

`computeFeatures` *Get all main features in ORFik*

---

### Description

If you want to get all the NGS and/or sequence 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 (in the "see also" section below) to see all of them. Example, if you want to know what the "te" column is, check out: ?translationalEff.
A short description of each feature is also shown here:

** **NGS features** ** If not stated otherwise stated, the feature apply to Ribo-seq.

- countRFP : raw counts of Ribo-seq
- fpkmRFP : FPKM
- fpkmRNA : FPKM of RNA-seq
- te : Translation efficiency Ribo-seq / RNA-seq FPKM

- floss : Fragment length similarity score
- entropyRFP : Positional entropy
- disengagementScores : downstream coverage from ORF
- RRS: Ribosome release score
- RSS: Ribosome staling score
- ORFScores: Periodicity score, does frame 0 have more reads
- ioScore: inside outside score: coverage ORF / coverage rest of transcript
- startCodonCoverage: Coverage over start codon + 2nt before start codon
- startRegionCoverage: Coverage over codon 2 & 3
- startRegionRelative: Peakness of TIS, startCodonCoverage / startRegionCoverage, 0-n

** **Sequence features** **

- kozak : Similarity to kozak sequence for organism score, 0-1
- gc : GC percentage, 0-1
- StartCodons : Start codon as a string, "ATG"
- StopCodons : stop codon as a string, "TAA"
- fractionLengths : ORF length compared to transcript, 0-1

** **uORF features** **

- distORFCDS : Distance from ORF stop site to CDS, -n:n
- inFrameCDS : Is ORF in frame with downstream CDS, T/F
- isOverlappingCds : Is ORF overlapping with downstream CDS, T/F
- rankInTx : ORF with most upstream start codon is 1, 1-n

## Usage

```
computeFeatures(
  grl,
  RFP,
  RNA = NULL,
  Gtf,
  faFile = NULL,
  riboStart = 26,
  riboStop = 34,
  sequenceFeatures = TRUE,
  uorfFeatures = TRUE,
  grl.is.sorted = FALSE,
  weight.RFP = 1L,
  weight.RNA = 1L
)
```

## Arguments

| | |
|---|---|
| grl | a GRangesList object with usually ORFs, but can also be either leaders, cds', 3' utrs, etc. This is the regions you want to score. |
| RFP | RiboSeq reads as GAlignments , GRanges or GRangesList object |
| RNA | RnaSeq reads as GAlignments , GRanges or GRangesList object |
| Gtf | a TxDb object of a gtf file or path to gtf, gff .sqlite etc. |
| faFile | a path to fasta indexed genome, an open FaFile, a BSgenome, or path to ORFik experiment with valid genome. |
| riboStart | usually 26, the start of the floss interval, see ?floss |
| riboStop | usually 34, the end of the floss interval |
| sequenceFeatures | a logical, default TRUE, include all sequence features, that is: Kozak, fractionLengths, distORFCDS, isInFrame, isOverlapping and rankInTx. uorfFeatures = FALSE will remove the 4 last. |
| uorfFeatures | a logical, default TRUE, include all uORF sequence features, that is: distORFCDS, isInFrame, isOverlapping and rankInTx |
| grl.is.sorted | logical (F), a speed up if you know argument grl is sorted, set this to TRUE. |
| weight.RFP | a vector (default: 1L). Can also be character name of column in RFP. As in translationalEff(weight = "score") for: GRanges("chr1", 1, "+", score = 5), would mean score column tells that this alignment region was found 5 times. |
| weight.RNA | Same as weightRFP but for RNA weights. (default: 1L) |

## Details

If you used CageSeq to reannotate your leaders, your txDB object must contain the reassigned leaders. Use [reassignTxDbByCage()] to get the txdb.
As a note the library is reduced to only reads overlapping 'tx', so the library size in fpkm calculation is done on this subset. This will help remove rRNA and other contaminants.
Also if you have only unique reads with a weight column, explaining the number of duplicated reads, set weights to make calculations correct. See getWeights

## 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(), countOverlapsW(), disengagementScore(), distToCds(), distToTSS(), entropy(), floss(), fpkm(), fpkm_calc(), fractionLength(), initiationScore(), insideOutsideORF(), isInFrame(), isOverlapping(), kozakSequenceScore(), orfScore(), rankOrder(), ribosomeReleaseScore(), ribosomeStallingScore(), startRegion(), startRegionCoverage(), stopRegion(), subsetCoverage(), translationalEff()

## Examples

```
# Here we make an example from scratch
# Usually the ORFs are found in orfik, which makes names for you etc.
gtf <- system.file("extdata/references/danio_rerio", "annotations.gtf",
 package = "ORFik") ## location of the gtf file

suppressWarnings(txdb <- loadTxdb(gtf))
# use cds' as ORFs for this example
ORFs <- loadRegion(txdb, "cds")
ORFs <- makeORFNames(ORFs) # need ORF names
# make Ribo-seq data,
RFP <- unlistGrl(firstExonPerGroup(ORFs))
computeFeatures(ORFs, RFP, Gtf = txdb)
# For more details see vignettes.
```

---

`computeFeaturesCage` *Get all main features in ORFik*

---

## Description

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

## Usage

```
computeFeaturesCage(
  grl,
  RFP,
  RNA = NULL,
  Gtf = NULL,
  tx = NULL,
  fiveUTRs = NULL,
  cds = NULL,
  threeUTRs = NULL,
  faFile = NULL,
  riboStart = 26,
  riboStop = 34,
  sequenceFeatures = TRUE,
  uorfFeatures = TRUE,
  grl.is.sorted = FALSE,
  weight.RFP = 1L,
  weight.RNA = 1L
)
```

## Arguments

| | |
|---|---|
| `grl` | a `GRangesList` object with usually ORFs, but can also be either leaders, cds', 3' utrs, etc. This is the regions you want to score. |

| | |
|---|---|
| RFP | RiboSeq reads as GAlignments , GRanges or GRangesList object |
| RNA | RnaSeq reads as GAlignments , GRanges or GRangesList object |
| Gtf | a TxDb object of a gtf file or path to gtf, gff .sqlite etc. |
| tx | a GRangesList of transcripts, normally called from: exonsBy(Gtf, by = "tx", use.names = T) only add this if you are not including Gtf file If you are using CAGE, 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 path to fasta indexed genome, an open FaFile, a BSgenome, or path to ORFik experiment with valid genome. |
| riboStart | usually 26, the start of the floss interval, see ?floss |
| riboStop | usually 34, the end of the floss interval |
| sequenceFeatures | a logical, default TRUE, include all sequence features, that is: Kozak, fractionLengths, distORFCDS, isInFrame, isOverlapping and rankInTx. uorfFeatures = FALSE will remove the 4 last. |
| uorfFeatures | a logical, default TRUE, include all uORF sequence features, that is: distORFCDS, isInFrame, isOverlapping and rankInTx |
| grl.is.sorted | logical (F), a speed up if you know argument grl is sorted, set this to TRUE. |
| weight.RFP | a vector (default: 1L). Can also be character name of column in RFP. As in translationalEff(weight = "score") for: GRanges("chr1", 1, "+", score = 5), would mean score column tells that this alignment region was found 5 times. |
| weight.RNA | Same as weightRFP but for RNA weights. (default: 1L) |

## Details

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

## Value

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

## See Also

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

## Examples

```
# a small example without cage-seq data:
# we will find ORFs in the 5' utrs
# and then calculate features on them

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
 tx_seqs <- 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))

 #ORFik:::computeFeaturesCage(grl = fiveUTR_ORFs, RFP = RFP,
 #  RNA = RNA, Gtf = txdb, faFile = faFile)

}
# See vignettes for more examples
```

---

`config` *Read directory config for ORFik experiments*

---

## Description

Defines a folder for: 1. fastq files (raw data)
2. bam files (processed data)
3. references (organism annotation and STAR index)
4. experiments (Location to store and load all `experiment` .csv files) Update or use another config using `config.save()` function.

### Usage

```
config(
  file = config_file(old_config_location = old_config_location),
  old_config_location = "~/Bio_data/ORFik_config.csv"
)
```

### Arguments

| | |
|---|---|
| file | location of config csv, default: config_file(old_config_location = old_config_location) |
| old_config_location | path, old config location before BiocFileCache implementation. Will copy this to cache directory and delete old version. This is done to follow bioc rules on not writing to user home directory. |

### Value

a named character vector of length 3

### Examples

```
## Make with default config path
#config()
```

---

## config.exper *Set directories for experiment*

---

### Description

Defines a folder for: 1. fastq files (raw_data)
2. bam files (processed data)
3. references (organism annotation and STAR index)
4. Experiment (name of experiment)

### Usage

```
config.exper(experiment, assembly, type, config = ORFik::config())
```

### Arguments

| | |
|---|---|
| experiment | short name of experiment (must be valid as a folder name) |
| assembly | name of organism and assembly (must be valid as a folder name) |
| type | name of sequencing type, Ribo-seq, RNA-seq, CAGE.. Can be more than one. |
| config | a named character vector of length 3, default: ORFik::config() |

### Value

named character vector of paths for experiment

## Examples

```
# Where should files go in general?
ORFik::config()
# Paths for project: "Alexaki_Human" containing Ribo-seq and RNA-seq:
#config.exper("Alexaki_Human", "Homo_sapiens_GRCh38_101", c("Ribo-seq", "RNA-seq"))
```

---

`config.save` *Save/update directory config for ORFik experiments*

---

## Description

Defines a folder for fastq files (raw_data), bam files (processed data) and references (organism annotation and STAR index)

## Usage

```
config.save(
  file = config_file(),
  fastq.dir = file.path(base.dir, "raw_data"),
  bam.dir = file.path(base.dir, "processed_data"),
  reference.dir = file.path(base.dir, "references"),
  exp.dir = file.path(base.dir, "ORFik_experiments/"),
  base.dir = "~/Bio_data",
 conf = data.frame(type = c("fastq", "bam", "ref", "exp"), directory = c(fastq.dir,
    bam.dir, reference.dir, exp.dir))
)
```

## Arguments

| | |
|---|---|
| `file` | location of config csv, default: config_file(old_config_location = old_config_location) |
| `fastq.dir` | directory where ORFik puts fastq file directories, default: file.path(base.dir, "raw_data"), which is retrieved with: `config()["fastq"]` |
| `bam.dir` | directory where ORFik puts bam file directories, default: file.path(base.dir, "processed_data"), which is retrieved with: `config()["bam"]` |
| `reference.dir` | directory where ORFik puts reference file directories, default: file.path(base.dir, "references"), which is retrieved with: `config()["ref"]` |
| `exp.dir` | directory where ORFik puts experiment csv files, default: file.path(base.dir, "ORFik_experiments/"), which is retrieved with: `config()["exp"]` |
| `base.dir` | base directory for all output directories, default: "~/Bio_data" |
| `conf` | data.frame of complete conf object, default: data.frame(type = c("fastq", "bam", "ref", "exp"), directory = c(fastq.dir, bam.dir, reference.dir, exp.dir)) |

## Value

invisible(NULL), file saved to disc

## Examples

```
# Overwrite default config, with new base directory for files
#config.save(base.dir = "/media/Bio_data/") # Output files go here instead
# of ~/Bio_data
## Dont do this, but for understanding here is how to make a second config
#new_config_path <- config_file(query = "ORFik_config_2")
#config.save(new_config_path, "/media/Bio_data/raw_data/",
# "/media/Bio_data/processed_data", /media/Bio_data/references/)
```

---

`config_file` *Get path for ORFik config in cache*

---

## Description

Get path for ORFik config in cache

## Usage

```
config_file(
  cache = BiocFileCache::getBFCOption("CACHE"),
  query = "ORFik_config",
  ask = interactive(),
  old_config_location = "~/Bio_data/ORFik_config.csv"
)
```

## Arguments

| | |
|---|---|
| `cache` | path to bioc cache directory with rname from query argument. Default is: `BiocFileCache::getBFCOption("CACHE")` For info, see: [BiocFileCache::BiocFileCache()] |
| `query` | default: "ORFik_config". Exact rname of the file in cache. |
| `ask` | logical, default interactive(). |
| `old_config_location` | path, old config location before BiocFileCache implementation. Will copy this to cache directory and delete old version. This is done to follow bioc rules on not writing to user home directory. |

## Value

a file path in cache

## Examples

```
config_file()
# Another config path
config_file(query = "ORFik_config_2")
```

convertLibs *Converted format of NGS libraries*

## Description

Export as either .ofst, .wig, .bigWig,.bedo (legacy format) or .bedoc (legacy format) files:
Export files as .ofst for fastest load speed into R.
Export files as .wig / bigWig for use in IGV or other genome browsers.
The input files are checked if they exist from: `envExp(df)`.

## Usage

```
convertLibs(
  df,
  out.dir = libFolder(df),
  addScoreColumn = TRUE,
  addSizeColumn = TRUE,
  must.overlap = NULL,
  method = "None",
  type = "ofst",
  input.type = "ofst",
  reassign.when.saving = FALSE,
  envir = envExp(df),
  force = TRUE,
  library.names = bamVarName(df),
 libs = outputLibs(df, type = input.type, chrStyle = must.overlap, library.names =
    library.names, output.mode = "list", force = force, BPPARAM = BPPARAM),
  BPPARAM = bpparam()
)
```

## Arguments

| | |
|---|---|
| df | an ORFik experiment |
| out.dir | optional output directory, default: libFolder(df), if it is NULL, it will just reassign R objects to simplified libraries. Will then create a final folder specfied as: paste0(out.dir, "/", type, "/"). Here the files will be saved in format given by the type argument. |
| addScoreColumn | logical, default TRUE, if FALSE will not add replicate numbers as score column, see ORFik::convertToOneBasedRanges. |
| addSizeColumn | logical, default TRUE, if FALSE will not add size (width) as size column, see ORFik::convertToOneBasedRanges. Does not apply for (GAlignment version of.ofst) or .bedoc. Since they contain the original cigar. |
| must.overlap | default (NULL), else a GRanges / GRangesList object, so only reads that overlap (must.overlap) are kept. This is useful when you only need the reads over transcript annotation or subset etc. |

| | |
|---|---|
| method | character, default "None", the method to reduce ranges, for more info see convertToOneBasedRanges |
| type | character, output format, default "ofst". Alternatives: "ofst", "bigWig", "wig","bedo" or "bedoc". Which format you want. Will make a folder within out.dir with this name containing the files. |
| input.type | character, input type "ofst". Remember this function uses the loaded libraries if existing, so this argument is usually ignored. Only used if files do not already exist. |
| reassign.when.saving | logical, default FALSE. If TRUE, will reassign library to converted form after saving. Ignored when out.dir = NULL. |
| envir | environment to save to, default envExp(df), which defaults to .GlobalEnv, but can be set with envExp(df) <- new.env() etc. |
| force | logical, default TRUE If TRUE, reload library files even if matching named variables are found in environment used by experiment (see envExp) A simple way to make sure correct libraries are always loaded. FALSE is faster if data is loaded correctly already. |
| libs | list, output of outputLibs as list of GRanges/GAlignments/GAlignmentPairs objects. Set input.type and force arguments to define parameters. |
| BPPARAM | how many cores/threads to use? default: bpparam(). To see number of threads used, do bpparam()$workers. You can also add a time remaining bar, for a more detailed pipeline. |

## Details

We advice you to not use this directly, as other function are more safe for library type conversions. See family description below. This is mostly used internally in ORFik. It is only adviced to use if large bam files are already loaded in R and conversions are wanted from those.

See export.ofst, export.wiggle, export.bedo and export.bedoc for information on file formats.
If libraries of the experiment are already loaded into environment (default: .globalEnv) is will export using those files as templates. If they are not in environment the .ofst files from the bam files are loaded (unless you are converting to .ofst then the .bam files are loaded).

## Value

invisible NULL (saves files to disc or R .GlobalEnv)

## See Also

Other lib_converters: convert_bam_to_ofst(), convert_to_bigWig(), convert_to_covRle(), convert_to_covRleList()

## Examples

```
df <- ORFik.template.experiment()
#convertLibs(df, out.dir = NULL)
# Keep only 5' ends of reads
#convertLibs(df, out.dir = NULL, method = "5prime")
```

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

## Description

There are 5 ways of doing this
1. Take 5' ends, reduce away rest (5prime)
2. Take 3' ends, reduce away rest (3prime)
3. Tile to 1-mers and include all (tileAll)
4. Take middle point per GRanges (middle)
5. Get original with metacolumns (None)
You can also do multiple at a time, then output is GRangesList, where each list group is the operation (5prime is [1], 3prime is [2] etc)
Many other ways to do this have their own functions, like startSites and stopSites etc. To retain information on original width, set addSizeColumn to TRUE. To compress data, 1 GRanges object per unique read, set addScoreColumn to TRUE. This will give you a score column with how many duplicated reads there were in the specified region.

## Usage

```
convertToOneBasedRanges(
  gr,
  method = "5prime",
  addScoreColumn = FALSE,
  addSizeColumn = FALSE,
  after.softclips = TRUE,
  along.reference = FALSE,
  reuse.score.column = TRUE
)
```

## Arguments

| | |
|---|---|
| `gr` | GRanges, GAlignment or GAlignmentPairs object to reduce. |
| `method` | character, default `"5prime"`, the method to reduce ranges, see NOTE for more info. |
| `addScoreColumn` | logical (FALSE), if TRUE, add a score column that sums up the hits per unique range. This will make each read unique, so that each read is 1 time, and score column gives the number of collapsed hits. A useful compression. If addSizeColumn is FALSE, it will not differentiate between reads with same start and stop, but different length. If addSizeColumn is FALSE, it will remove it. Collapses after conversion. |
| `addSizeColumn` | logical (FALSE), if TRUE, add a size column that for each read, that gives original width of read. Useful if you need original read lengths. This takes care of soft clips etc. If collapsing reads, each unique range will be grouped also by size. |

`after.softclips`
: logical (TRUE), include softclips in width. Does not apply if along.reference is TRUE.

`along.reference`
: logical (FALSE), example: The cigar "26MI2" is by default width 28, but if along.reference is TRUE, it will be 26. The length of the read along the reference. Also "1D20M" will be 21 if by along.reference is TRUE. Intronic regions (cigar: N) will be removed. So: "1M200N19M" is 20, not 220.

`reuse.score.column`
: logical (TRUE), if addScoreColumn is TRUE, and a score column exists, will sum up the scores to create a new score. If FALSE, will skip old score column and create new according to number of replicated reads after conversion. If addScoreColumn is FALSE, this argument is ignored.

## Details

NOTE: Note: For cigar based ranges (GAlignments), the 5' end is the first non clipped base (neither soft clipped or hard clipped from 5'). This is following the default of bioconductor. For special case of GAlignmentPairs, 5prime will only use left (first) 5' end and read and 3prime will use only right (last) 3' end of read in pair. tileAll and middle can possibly find poinst that are not in the reads since: lets say pair is 1-5 and 10-15, middle is 7, which is not in the read.

## Value

Converted GRanges object

## See Also

Other utils: `bedToGR()`, `export.bed12()`, `export.bigWig()`, `export.fstwig()`, `export.wiggle()`, `fimport()`, `findFa()`, `fread.bed()`, `optimizeReads()`, `readBam()`, `readBigWig()`, `readWig()`

## Examples

```
gr <- GRanges("chr1", 1:10,"+")
# 5 prime ends
convertToOneBasedRanges(gr)
# is equal to convertToOneBasedRanges(gr, method = "5prime")
# 3 prime ends
convertToOneBasedRanges(gr, method = "3prime")
# With lengths
convertToOneBasedRanges(gr, addSizeColumn = TRUE)
# With score (# of replicates)
gr <- rep(gr, 2)
convertToOneBasedRanges(gr, addSizeColumn = TRUE, addScoreColumn = TRUE)
```

`convert_bam_to_ofst` *Convert libraries to ofst*

## Description

Saved by default in folder "ofst" relative to default libraries of experiment. Speeds up loading of full files compared to bam by large margins.

## Usage

```
convert_bam_to_ofst(
  df,
  in_files = filepath(df, "default"),
  out_dir = file.path(libFolder(df), "ofst"),
  verbose = TRUE,
  strandMode = rep(0, length(in_files))
)
```

## Arguments

| | |
|---|---|
| `df` | an ORFik `experiment`, or NULL is allowed if both in_files and out_dir is specified manually. |
| `in_files` | paths to input files, default: `filepath(df, "default")` with bam format files. |
| `out_dir` | paths to output files, default `file.path(libFolder(df), "cov_RLE")`. |
| `verbose` | logical, default TRUE, message about library output status. |
| `strandMode` | numeric, default 0. Only used for paired end bam files. One of (0: strand = *, 1: first read of pair is +, 2: first read of pair is -). See ?strandMode. Note: Sets default to 0 instead of 1, as readGAlignmentPairs uses 1. This is to guarantee hits, but will also make mismatches of overlapping transcripts in opposite directions. |

## Details

If you want to keep bam files loaded or faster conversion if you already have them loaded, use ORFik::convertLibs instead

## Value

invisible(NULL), files saved to disc

## See Also

Other lib_converters: `convertLibs()`, `convert_to_bigWig()`, `convert_to_covRle()`, `convert_to_covRleList()`

## Examples

```
df <- ORFik.template.experiment.zf()
## Usually do default folder, here we use tmpdir
folder_to_save <- file.path(tempdir(), "ofst")
convert_bam_to_ofst(df, out_dir = folder_to_save)
fimport(file.path(folder_to_save, "ribo-seq.ofst"))
```

---

# convert_to_bigWig *Convert to BigWig*

---

## Description

Convert to BigWig

## Usage

```
convert_to_bigWig(
  df,
  in_files = filepath(df, "pshifted"),
  out_dir = file.path(libFolder(df), "bigwig"),
  split.by.strand = TRUE,
  split.by.readlength = FALSE,
  seq_info = seqinfo(df),
  weight = "score",
  is_pre_collapsed = FALSE,
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| df | an ORFik experiment, or NULL is allowed if both in_files and out_dir is specified manually. |
| in_files | paths to input files, default pshifted files: `filepath(df, "pshifted")` in ofst format |
| out_dir | paths to output files, default `file.path(libFolder(df), "bigwig")`. |
| split.by.strand | logical, default TRUE, split into forward and reverse strand RleList inside covRle object. |
| split.by.readlength | logical, default FALSE, split into files for each readlength, defined by readWidths(x) for each file. |
| seq_info | SeqInfo object, default `seqinfo(findFa(df))` |
| weight | integer, numeric or single length character. Default "score". Use score column in loaded in_files. |

is_pre_collapsed
: logical, default FALSE. Have you already collapsed reads with collapse.by.scores, so each positions is only in 1 GRanges object with a score column per readlength? Set to TRUE, only if you are sure, will give a speedup.

verbose
: logical, default TRUE, message about library output status.

## Value

invisible(NULL), files saved to disc

## See Also

Other lib_converters: `convertLibs()`, `convert_bam_to_ofst()`, `convert_to_covRle()`, `convert_to_covRleList()`

## Examples

```
df <- ORFik.template.experiment()[10,]
## Usually do default folder, here we use tmpdir
folder_to_save <- file.path(tempdir(), "bigwig")
convert_to_bigWig(df, out_dir = folder_to_save)
fimport(file.path(folder_to_save, c("RFP_Mutant_rep2_forward.bigWig",
 "RFP_Mutant_rep2_reverse.bigWig")))
```

---

# convert_to_covRle *Convert libraries to covRle*

---

## Description

Saved by default in folder "cov_RLE" relative to default libraries of experiment

## Usage

```
convert_to_covRle(
  df,
  in_files = filepath(df, "pshifted"),
  out_dir = file.path(libFolder(df), "cov_RLE"),
  split.by.strand = TRUE,
  split.by.readlength = FALSE,
  seq_info = seqinfo(df),
  weight = "score",
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| df | an ORFik experiment, or NULL is allowed if both in_files and out_dir is specified manually. |
| in_files | paths to input files, default pshifted files: filepath(df, "pshifted") in ofst format |
| out_dir | paths to output files, default file.path(libFolder(df), "cov_RLE"). |
| split.by.strand | logical, default TRUE, split into forward and reverse strand RleList inside covRle object. |
| split.by.readlength | logical, default FALSE, split into files for each readlength, defined by readWidths(x) for each file. |
| seq_info | SeqInfo object, default seqinfo(findFa(df)) |
| weight | integer, numeric or single length character. Default "score". Use score column in loaded in_files. |
| verbose | logical, default TRUE, message about library output status. |

## Value

invisible(NULL), files saved to disc

## See Also

Other lib_converters: convertLibs(), convert_bam_to_ofst(), convert_to_bigWig(), convert_to_covRleList()

## Examples

```
df <- ORFik.template.experiment()[10,]
## Usually do default folder, here we use tmpdir
folder_to_save <- file.path(tempdir(), "cov_RLE")
convert_to_covRle(df, out_dir = folder_to_save)
fimport(file.path(folder_to_save, "RFP_Mutant_rep2.covrds"))
```

---

# convert_to_covRleList *Convert libraries to covRleList objects*

---

## Description

Useful to store reads separated by readlength, for much faster coverage calculation. Saved by default in folder "cov_RLE_List" relative to default libraries of experiment

## Usage

```
convert_to_covRleList(
  df,
  in_files = filepath(df, "pshifted"),
  out_dir = file.path(libFolder(df), "cov_RLE_List"),
  out_dir_merged = file.path(libFolder(df), "cov_RLE"),
  split.by.strand = TRUE,
  seq_info = seqinfo(df),
  weight = "score",
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| df | an ORFik experiment, or NULL is allowed if both in_files and out_dir is specified manually. |
| in_files | paths to input files, default pshifted files: filepath(df, "pshifted") in ofst format |
| out_dir | paths to output files, default file.path(libFolder(df), "cov_RLE_List"). |
| out_dir_merged | character vector of paths, default: file.path(libFolder(df), "cov_RLE"). Paths to merged output files, Set to NULL to skip making merged covRle. |
| split.by.strand | logical, default TRUE, split into forward and reverse strand RleList inside covRle object. |
| seq_info | SeqInfo object, default seqinfo(findFa(df)) |
| weight | integer, numeric or single length character. Default "score". Use score column in loaded in_files. |
| verbose | logical, default TRUE, message about library output status. |

## Value

invisible(NULL), files saved to disc

## See Also

Other lib_converters: convertLibs(), convert_bam_to_ofst(), convert_to_bigWig(), convert_to_covRle()

## Examples

```
df <- ORFik.template.experiment()[10,]
## Usually do default folder, here we use tmpdir
folder_to_save <- file.path(tempdir(), "cov_RLE_List")
folder_to_save_merged <- file.path(tempdir(), "cov_RLE")
ORFik:::convert_to_covRleList(df, out_dir = folder_to_save,
out_dir_merged = folder_to_save_merged)
fimport(file.path(folder_to_save, "RFP_Mutant_rep2.covrds"))
```

`convert_to_fstWig` *Convert to fstwig*

## Description

Will split files by chromosome for faster loading for now. This feature might change in the future!

## Usage

```
convert_to_fstWig(
  df,
  in_files = filepath(df, "pshifted"),
  out_dir = file.path(libFolder(df), "fstwig"),
  split.by.strand = TRUE,
  split.by.readlength = FALSE,
  seq_info = seqinfo(df),
  weight = "score",
  is_pre_collapsed = FALSE,
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| `df` | an ORFik `experiment`, or NULL is allowed if both in_files and out_dir is specified manually. |
| `in_files` | paths to input files, default pshifted files: `filepath(df, "pshifted")` in ofst format |
| `out_dir` | paths to output files, default `file.path(libFolder(df), "bigwig")`. |
| `split.by.strand` | logical, default TRUE, split into forward and reverse strand RleList inside covRle object. |
| `split.by.readlength` | logical, default FALSE, split into files for each readlength, defined by readWidths(x) for each file. |
| `seq_info` | SeqInfo object, default `seqinfo(findFa(df))` |
| `weight` | integer, numeric or single length character. Default "score". Use score column in loaded in_files. |
| `is_pre_collapsed` | logical, default FALSE. Have you already collapsed reads with collapse.by.scores, so each positions is only in 1 GRanges object with a score column per readlength? Set to TRUE, only if you are sure, will give a speedup. |
| `verbose` | logical, default TRUE, message about library output status. |

## Value

invisible(NULL), files saved to disc

`correlation.plots` *Correlation plots between all samples*

## Description

Get correlation plot of raw counts and/or log2(count + 1) over selected region in: c("mrna", "leaders", "cds", "trailers")

Note on correlation: Pearson correlation, using pairwise observations to fill in NA values for the covariance matrix.

## Usage

```
correlation.plots(
  df,
  output.dir,
  region = "mrna",
  type = "fpkm",
  height = 400,
  width = 400,
  size = 0.15,
  plot.ext = ".pdf",
  complex.correlation.plots = TRUE,
  data_for_pairs = countTable(df, region, type = type),
  as_gg_list = FALSE,
  text_size = 4,
  method = c("pearson", "spearman")[1]
)
```

## Arguments

| | |
|---|---|
| `df` | an ORFik `experiment` |
| `output.dir` | directory to save to, named : cor_plot, cor_plot_log2 and/or cor_plot_simple with either .pdf or .png |
| `region` | a character (default: mrna), make raw count matrices of whole mrnas or one of (leaders, cds, trailers) |
| `type` | which value to use, "fpkm", alternative "counts". |
| `height` | numeric, default 400 (in mm) |
| `width` | numeric, default 400 (in mm) |
| `size` | numeric, size of dots, default 0.15. Deprecated. |
| `plot.ext` | character, default: ".pdf". Alternatives: ".png" or ".jpg". |
| `complex.correlation.plots` | logical, default TRUE. Add in addition to simple correlation plot two computationally heavy dots + correlation plots. Useful for deeper analysis, but takes longer time to run, especially on low-quality gpu computers. Set to FALSE to skip these. |

| | |
|---|---|
| data_for_pairs | a data.table from ORFik::countTable of counts wanted. Default is fpkm of all mRNA counts over all libraries. |
| as_gg_list | logical, default FALSE. Return as a list of ggplot objects instead of as a grob. Gives you the ability to modify plots more directly. |
| text_size | size of correlation numbers |
| method | c("pearson", "spearman")[1] |

## Value

invisible(NULL) / if as_gg_list is TRUE, return a list of raw plots.

---

`cor_plot` *Get correlation between columns*

---

## Description

Get correlation between columns

## Usage

```
cor_plot(
  dt_cor,
  col = c(low = "blue", high = "red", mid = "white", na.value = "white"),
  limit = c(ifelse(min(dt_cor$Cor, na.rm = TRUE) < 0, -1, 0), 1),
  midpoint = mean(limit),
  label_name = "Pearson\nCorrelation",
  text_size = 4,
  legend.position = c(0.4, 0.7),
  legend.direction = "horizontal"
)
```

## Arguments

| | |
|---|---|
| dt_cor | a data.table, with column Cor |
| col | colors c(low = "blue", high = "red", mid = "white", na.value = "white") |
| limit | default (-1, 1), defined by: `c(ifelse(min(dt_cor$Cor, na.rm = TRUE) < 0, -1, 0), 1)` |
| midpoint | midpoint of correlation values in label coloring. |
| label_name | name of correlation method, default `"Pearson Correlation"` with newline after Pearson. |
| text_size | size of correlation numbers |
| legend.position | default c(0.4, 0.7), other: "top", "right",.. |
| legend.direction | default "horizontal", or "vertical" |

## Value

a ggplot (heatmap)

---

`cor_table` *Get correlation between columns*

---

## Description

Get correlation between columns

## Usage

```
cor_table(
  dt,
  method = c("pearson", "spearman")[1],
  upper_triangle = TRUE,
  decimals = 2,
  melt = TRUE,
  na.rm.melt = TRUE
)
```

## Arguments

| | |
|---|---|
| `dt` | a data.table |
| `method` | c("pearson", "spearman")[1] |
| `upper_triangle` | logical, default TRUE. Make lower triangle values NA. |
| `decimals` | numeric, default 2. How many decimals for correlation |
| `melt` | logical, default TRUE. |
| `na.rm.melt` | logical, default TRUE. Remove NA values from melted table. |

## Value

a data.table with 3 columns, Var1, Var2 and Cor

countOverlapsW *CountOverlaps with weights*

## Description

Similar to countOverlaps, but takes an optional weight column. This is usually the score column

## Usage

```
countOverlapsW(query, subject, weight = NULL, ...)
```

## Arguments

| | |
|---|---|
| query | IRanges, IRangesList, GRanges, GRangesList object. Usually transcript a transcript region. |
| subject | GRanges, GRangesList, GAlignment, usually reads. |
| weight | (default: NULL), if defined either numeric or character name of valid meta column in subject. If weight is single numeric, it is used for all. A normall weight is the score column given as weight = "score". GRanges("chr1", 1, "+", score = 5), would mean score column tells that this alignment region was found 5 times. |
| ... | additional arguments passed to countOverlaps/findOverlaps |

## Value

a named vector of number of overlaps to subject weigthed by 'weight' column.

## See Also

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

## Examples

```
gr1 <- GRanges(seqnames="chr1",
               ranges=IRanges(start = c(4, 9, 10, 30),
                              end = c(4, 15, 20, 31)),
               strand="+")
gr2 <- GRanges(seqnames="chr1",
               ranges=IRanges(start = c(1, 4, 15, 25),
                              end = c(2, 4, 20, 26)),
               strand=c("+"),
               score=c(10, 20, 15, 5))
countOverlaps(gr1, gr2)
countOverlapsW(gr1, gr2, weight = "score")
```

countTable *Extract count table directly from experiment*

## Description

Used to quickly load pre-created read count tables to R.
If df is experiment: Extracts by getting /QC_STATS directory, and searching for region Requires ORFikQC to have been run on experiment, to get default count tables!

## Usage

```
countTable(
  df,
  region = "mrna",
  type = "count",
  collapse = FALSE,
  count.folder = "default"
)
```

## Arguments

| | |
|---|---|
| df | an ORFik experiment or path to folder with countTable, use path if not same folder as experiment libraries. Will subset to the count tables specified if df is experiment. If experiment has 4 rows and you subset it to only 2, then only those 2 count tables will be outputted. |
| region | a character vector (default: "mrna"), make raw count matrices of whole mrnas or one of (leaders, cds, trailers). |
| type | character, default: "count" (raw counts matrix). Which object type and normalization do you want ? "summarized" (SummarizedExperiment object), "deseq" (Deseq2 experiment, design will be all valid non-unique columns except replicates, change by using DESeq2::design, normalization alternatives are: "fpkm", "log2fpkm" or "log10fpkm". |
| collapse | a logical/character (default FALSE), if TRUE all samples within the group SAMPLE will be collapsed to one. If "all", all groups will be merged into 1 column called merged_all. Collapse is defined as rowSum(elements_per_group) / ncol(elements_per_group) |
| count.folder | character, default "auto" (Use count tables from original bam files stored in "QC_STATS", these are like HTseq count tables). To load your custome count tables from pshifted reads, set to "pshifted" (remember to create the pshifted tables first!). If you have custom ranges, like reads over uORFs stored in a folder called "/uORFs" relative to the bam files, set to "uORFs". Always create these custom count tables with makeSummarizedExperimentFromBam. Always make the location of the folder directly inside the bam file directory! |

## Details

If df is path to folder: Loads the the file in that directory with the regex region.rds, where region is what is defined by argument, if multiple exist, see if any start with "countTable_", if so, subset. If loaded as SummarizedExperiment or deseq, the colData will be made from ORFik.experiment information.

## Value

a data.table/SummarizedExperiment/DESeq object of columns as counts / normalized counts per library, column name is name of library. Rownames must be unique for now. Might change.

## See Also

Other countTable: `countTable_regions()`

## Examples

```
# Make experiment
df <- ORFik.template.experiment()
# Make QC report to get counts ++ (not needed for this template)
# ORFikQC(df)

# Get count Table of mrnas
# countTable(df, "mrna")
# Get count Table of cds
# countTable(df, "cds")
# Get count Table of mrnas as fpkm values
# countTable(df, "mrna", type = "count")
# Get count Table of mrnas with collapsed replicates
# countTable(df, "mrna", collapse = TRUE)
# Get count Table of mrnas as summarizedExperiment
# countTable(df, "mrna", type = "summarized")
# Get count Table of mrnas as DESeq2 object,
# for differential expression analysis
# countTable(df, "mrna", type = "deseq")
```

---

# countTable_regions *Make a list of count matrices from experiment*

---

## Description

By default will make count tables over mRNA, leaders, cds and trailers for all libraries in experiment. region

## Usage

```
countTable_regions(
  df,
  out.dir = libFolder(df),
  longestPerGene = FALSE,
  geneOrTxNames = "tx",
  regions = c("mrna", "leaders", "cds", "trailers"),
  type = "count",
  lib.type = "ofst",
  weight = "score",
  rel.dir = "QC_STATS",
  forceRemake = FALSE,
  library.names = bamVarName(df),
  BPPARAM = bpparam()
)
```

## Arguments

| | |
|---|---|
| df | an ORFik experiment |
| out.dir | character, output directory, default: resFolder(df). Will make a folder within this called "QC_STATS" with all results in this directory. Warning: If you assign not default path, you will have a hazzle to load files later. Much easier to load count tables, statistics, ++ later with default. Update resFolder of df instead if needed. |
| longestPerGene | a logical (default FALSE), if FALSE all transcript isoforms per gene. Ignored if "region" is not a character of either: "mRNA","tx", "cds", "leaders" or "trailers". |
| geneOrTxNames | a character vector (default "tx"), should row names keep trancript names ("tx") or change to gene names ("gene") |
| regions | a character vector, default: c("mrna", "leaders", "cds", "trailers"), make raw count matrices of whole regions specified. Can also be a custom GRangesList of for example uORFs or a subset of cds etc. |
| type | default: "count" (raw counts matrix), alternative is "fpkm", "log2fpkm" or "log10fpkm" |
| lib.type | a character(default: "default"), load files in experiment or some precomputed variant, either "ofst", "bedo", "bedoc" or "pshifted". These are made with ORFik:::convertLibs() or shiftFootprintsByExperiment(). Can also be custom user made folders inside the experiments bam folder. |
| weight | numeric or character, a column to score overlaps by. Default "score", will check for a metacolumn called "score" in libraries. If not found, will not use weights. |
| rel.dir | relative output directory for out.dir, default: "QC_STATS". For pshifted, write "pshifted". |
| forceRemake | logical, default FALSE. If TRUE, will not look for existing file count table files. |
| library.names | character, default: bamVarName(df). Names to load libraries as to environment and names to display in plots. |
| BPPARAM | how many cores/threads to use? default: bpparam() |

## Value

a list of data.table, 1 data.table per region. The regions will be the names the list elements.

## See Also

Other countTable: `countTable()`

## Examples

```
##Make experiment
df <- ORFik.template.experiment()
## Create count tables for all default regions
# countTable_regions(df)
## Pshifted reads (first create pshiftead libs)
# countTable_regions(df, lib.type = "pshifted", rel.dir = "pshifted")
```

---

`coverageByTranscriptC` *coverageByTranscript with coverage input*

---

## Description

Extends the function with direct genome coverage input, see `coverageByTranscript` for original function.

## Usage

```
coverageByTranscriptC(x, transcripts, ignore.strand = !strandMode(x))
```

## Arguments

| | |
|---|---|
| `x` | a covRle (one RleList for each strand in object), must have defined and correct seqlengths in its SeqInfo object. |
| `transcripts` | `GRangesList` |
| `ignore.strand` | a logical (default: length(x) == 1) |

## Value

Integer Rle of coverage, 1 per transcript

## coverageByTranscriptW *coverageByTranscript with weights*

### Description

Extends the function with weights, see `coverageByTranscript` for original function.

### Usage

```
coverageByTranscriptW(
  x,
  transcripts,
  ignore.strand = FALSE,
  weight = 1L,
  seqinfo.x.is.correct = FALSE
)
```

### Arguments

| | |
|---|---|
| `x` | reads (`GRanges`, `GAlignments`) |
| `transcripts` | `GRangesList` |
| `ignore.strand` | a logical (default: FALSE) |
| `weight` | a vector (default: 1L), if single number applies for all, else it must be the string name of a defined meta column in "x", that gives number of times a read was found. GRanges("chr1", 1, "+", score = 5), would mean score column tells that this alignment was found 5 times. |
| `seqinfo.x.is.correct` | logical, default FALSE. If you know x, has correct seqinfo, then you can save some computation time by setting this to TRUE. |

### Value

Integer Rle of coverage, 1 per transcript

## coverageHeatMap *Create a heatmap of coverage*

### Description

Creates a ggplot representing a heatmap of coverage:

- Rows : Position in region
- Columns : Read length

- Index intensity : (color) coverage scoring per index.

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

## Usage

```
coverageHeatMap(
  coverage,
  output = NULL,
  scoring = "zscore",
  legendPos = "right",
  addFracPlot = FALSE,
  xlab = "Position relative to start site",
  ylab = "Protected fragment length",
  colors = "default",
  title = NULL,
  increments.y = "auto",
  gradient.max = max(coverage$score)
)
```

## Arguments

| | |
|---|---|
| coverage | a data.table, e.g. output of scaledWindowCoverage |
| output | character string (NULL), if set, saves the plot as pdf or png to path given. If no format is given, is save as pdf. |
| scoring | character vector, default "zscore", Which scoring did you use to create? either of zscore, transcriptNormalized, sum, mean, median, .. see ?coverageScorings for info and more alternatives. |
| legendPos | a character, Default "right". Where should the fill legend be ? ("top", "bottom", "right", "left") |
| addFracPlot | Add margin histogram plot on top of heatmap with fractions per positions |
| xlab | the x-axis label, default "Position relative to start site" |
| ylab | the y-axis label, default "Protected fragment length" |
| colors | character vector, default: "default", this gives you: c("white", "yellow2", "yellow3", "lightblue", "blue", "navy"), do "high" for more high contrasts, or specify your own colors. |
| title | a character, default NULL (no title), what is the top title of plot? |
| increments.y | increments of y axis, default "auto". Or a numeric value < max position & > min position. |
| gradient.max | numeric, defualt: max(coverage$score). What data value should the top color be ? Good to use if you want to compare 2 samples, with the same color intensity, in that case set this value to the max score of the 2 coverage tables. |

## Details

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

- 0 reads in whole readlength :gray
- few reads in position :white
- medium reads in position :yellow
- many reads in position :dark blue

## Value

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

## See Also

Other heatmaps: `heatMapL()`, `heatMapRegion()`, `heatMap_single()`

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

## Examples

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

coverageHeatMap(coverage)

# With top sum bar
coverageHeatMap(coverage, addFracPlot = TRUE)
# See vignette for more examples
```

---

# coveragePerTiling *Get coverage per group*

---

## Description

It tiles each GRangesList group to width 1, and finds hits per position.
A range from 1:5 will split into c(1,2,3,4,5) and count hits on each. This is a safer speedup of coverageByTranscript from GenomicFeatures. It also gives the possibility to return as data.table, for faster computations.

## Usage

```
coveragePerTiling(
  grl,
  reads,
  is.sorted = FALSE,
  keep.names = TRUE,
  as.data.table = FALSE,
  withFrames = FALSE,
  weight = "score",
  drop.zero.dt = FALSE,
  fraction = NULL
)
```

## Arguments

| | |
|---|---|
| grl | a GRangesList of 5' utrs, CDS, transcripts, etc. |
| reads | a GAlignments, GRanges, or precomputed coverage as covRle (one for each strand) of RiboSeq, RnaSeq etc.<br>Weigths for scoring is default the 'score' column in 'reads'. Can also be random access paths to bigWig or fstwig file. Do not use random access for more than a few genes, then loading the entire files is usually better. File streaming is still in beta, so use with care! |
| is.sorted | logical (FALSE), is grl sorted. That is + strand groups in increasing ranges (1,2,3), and - strand groups in decreasing ranges (3,2,1) |
| keep.names | logical (TRUE), keep names or not. If as.data.table is TRUE, names (genes column) will be a factor column, if FALSE it will be an integer column (index of gene), so first input grl element is 1. Dropping names gives ~ 20 % speedup. If drop.zero.dt is FALSE, data.table will not return names, will use index (to avoid memory explosion). |
| as.data.table | a logical (FALSE), return as data.table with 2 columns, position and count. |
| withFrames | a logical (FALSE), only available if as.data.table is TRUE, return the ORF frame, 1,2,3, where position 1 is 1, 2 is 2 and 4 is 1 etc. |
| weight | (default: 'score'), if defined a character name of valid meta column in subject. GRanges("chr1", 1, "+", score = 5), would mean score column tells that this alignment region was found 5 times. ORFik ofst, bedoc and .bedo files contains a score column like this. As do CAGEr CAGE files and many other package formats. You can also assign a score column manually. |
| drop.zero.dt | logical FALSE, if TRUE and as.data.table is TRUE, remove all 0 count positions. This greatly speeds up and most importantly, greatly reduces memory usage. Will not change any plots, unless 0 positions are used in some sense. (mean, median, zscore coverage will only scale differently) |
| fraction | integer or character, a description column. Useful for grouping multiple outputs together. If returned as Rle, this is added as: metadata(coverage) <- list(fraction = fraction). If as.data.table it will be added as an additional column. |

### Details

NOTE: If reads contains a $score column, it will presume that this is the number of replicates per reads, weights for the coverage() function. So delete the score column or set weight to something else if this is not wanted.

### Value

a numeric RleList, one numeric-Rle per group with # of hits per position. Or data.table if as.data.table is TRUE, with column names c("count" [numeric or integer], "genes" [integer], "position" [integer])

### See Also

Other ExtendGenomicRanges: asTX(), extendLeaders(), extendTrailers(), reduceKeepAttr(), tile1(), txSeqsFromFa(), windowPerGroup()

### Examples

```
ORF <- GRanges(seqnames = "1",
               ranges = IRanges(start = c(1, 10, 20),
                                end = c(5, 15, 25)),
               strand = "+")
grl <- GRangesList(tx1_1 = ORF)
RFP <- GRanges("1", IRanges(25, 25), "+")
coveragePerTiling(grl, RFP, is.sorted = TRUE)
# now as data.table with frames
coveragePerTiling(grl, RFP, is.sorted = TRUE, as.data.table = TRUE,
                  withFrames = TRUE)
# With score column (usually replicated reads on that position)
RFP <- GRanges("1", IRanges(25, 25), "+", score = 5)
dt <- coveragePerTiling(grl, RFP, is.sorted = TRUE,
                        as.data.table = TRUE, withFrames = TRUE)
class(dt$count) # numeric
# With integer score column (faster and less space usage)
RFP <- GRanges("1", IRanges(25, 25), "+", score = 5L)
dt <- coveragePerTiling(grl, RFP, is.sorted = TRUE,
                        as.data.table = TRUE, withFrames = TRUE)
class(dt$count) # integer
```

---

## coverageScorings *Add a coverage scoring scheme*

---

### Description

Different scorings and groupings of a coverage representation.

### Usage

```
coverageScorings(coverage, scoring = "zscore", copy.dt = TRUE)
```

## Arguments

| | |
|---|---|
| coverage | a data.table containing at least columns (count, position), it is possible to have additionals: (genes, fraction, feature) |
| scoring | a character, one of (zscore, transcriptNormalized, mean, median, sum, log2sum, log10sum, sumLength, meanPos and frameSum, periodic, NULL). More info in details |
| copy.dt | logical TRUE, copy object, to avoid overwriting original object. Set to false to run function using reference to object, a speed up if original object is not needed. |

## Details

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

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

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

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

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

Given a data.table coverage of counts, add a scoring scheme. per: the grouping given, if genes is defined, group by per gene in default scoring.
Scorings:

- zscore (count-windowMean)/windowSD per)
- transcriptNormalized (sum(count / sum of counts per))
- mean (mean(count per))
- median (median(count per))
- sum (count per)
- log2sum (count per)
- log10sum (count per)
- sumLength (count per) / number of windows
- meanPos (mean per position per gene) used in scaledWindowPositions
- sumPos (sum per position per gene) used in scaledWindowPositions
- frameSum (sum per frame per gene) used in ORFScore
- frameSumPerL (sum per frame per read length)
- frameSumPerLG (sum per frame per read length per gene)
- fracPos (fraction of counts per position per gene)
- periodic (Fourier transform periodicity of meta coverage per fraction)
- NULL (no grouping, return input directly)

## Value

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

## See Also

Other coverage: `metaWindow()`, `regionPerReadLength()`, `scaledWindowPositions()`, `windowPerReadLength()`

## Examples

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

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

---

# coverage_to_dt *Convert coverage RleList to data.table*

---

## Description

Convert coverage RleList to data.table

## Usage

```
coverage_to_dt(
  coverage,
  keep.names = TRUE,
  withFrames = FALSE,
  weight = "score",
  drop.zero.dt = FALSE,
  fraction = NULL
)
```

## Arguments

| | |
|---|---|
| coverage | RleList with names |
| keep.names | logical (TRUE), keep names or not. If as.data.table is TRUE, names (genes column) will be a factor column, if FALSE it will be an integer column (index of gene), so first input grl element is 1. Dropping names gives ~ 20 % speedup. If drop.zero.dt is FALSE, data.table will not return names, will use index (to avoid memory explosion). |
| withFrames | a logical (FALSE), only available if as.data.table is TRUE, return the ORF frame, 1,2,3, where position 1 is 1, 2 is 2 and 4 is 1 etc. |

| | |
|---|---|
| weight | (default: 'score'), if defined a character name of valid meta column in subject. GRanges("chr1", 1, "+", score = 5), would mean score column tells that this alignment region was found 5 times. ORFik ofst, bedoc and .bedo files contains a score column like this. As do CAGEr CAGE files and many other package formats. You can also assign a score column manually. |
| drop.zero.dt | logical FALSE, if TRUE and as.data.table is TRUE, remove all 0 count positions. This greatly speeds up and most importantly, greatly reduces memory usage. Will not change any plots, unless 0 positions are used in some sense. (mean, median, zscore coverage will only scale differently) |
| fraction | integer or character, a description column. Useful for grouping multiple outputs together. If returned as Rle, this is added as: metadata(coverage) <- list(fraction = fraction). If as.data.table it will be added as an additional column. |

## Value

a data.table with column names c("count" [numeric or integer], "genes" [integer], "position" [integer])

---

# covRle *Coverage Rlelist for both strands*

---

## Description

Coverage Rlelist for both strands

## Usage

```
covRle(forward = RleList(), reverse = RleList())
```

## Arguments

| | |
|---|---|
| forward | a RleList with defined seqinfo for forward strand counts |
| reverse | a RleList with defined seqinfo for reverse strand counts |

## Value

a covRle object

## See Also

Other covRLE: covRle-class, covRleFromGR(), covRleList, covRleList-class

## Examples

```
covRle()
covRle(RleList(), RleList())
chr_rle <- RleList(chr1 = Rle(c(1,2,3), c(1,2,3)))
covRle(chr_rle, chr_rle)
```

## covRle-class *Coverage Rle for both strands or single*

### Description

Given a run of coverage(x) where x are reads, this class combines the 2 strands into 1 object

### Value

a covRLE object

### See Also

Other covRLE: covRle, covRleFromGR(), covRleList, covRleList-class

## covRleFromGR *Convert GRanges to covRle*

### Description

Convert GRanges to covRle

### Usage

```
covRleFromGR(x, weight = "AUTO", ignore.strand = FALSE)
```

### Arguments

| | |
|---|---|
| x | a GRanges, GAlignment or GAlignmentPairs object. Note that coverage calculation for GAlignment is slower, so usually best to call convertToOneBasedRanges on GAlignment object to speed it up. |
| weight | default "AUTO", pick 'score' column if exist, else all are 1L. Can also be a manually assigned meta column like 'score2' etc. |
| ignore.strand | logical, default FALSE. |

### Value

covRle object

### See Also

Other covRLE: covRle, covRle-class, covRleList, covRleList-class

## Examples

```
seqlengths <- as.integer(c(200, 300))
names(seqlengths) <- c("chr1", "chr2")
gr <- GRanges(seqnames = c("chr1", "chr1", "chr2", "chr2"),
              ranges = IRanges(start = c(10, 50, 100, 150), end = c(40, 80, 129, 179)),
              strand = c("+", "+", "-", "-"), seqlengths = seqlengths)
cov_both_strands <- covRleFromGR(gr)
cov_both_strands
cov_ignore_strand <- covRleFromGR(gr, ignore.strand = TRUE)
cov_ignore_strand
strandMode(cov_both_strands)
strandMode(cov_ignore_strand)
```

---

# covRleList *Coverage Rlelist for both strands*

---

## Description

Coverage Rlelist for both strands

## Usage

```
covRleList(list, fraction = names(list))
```

## Arguments

| | |
|---|---|
| `list` | a list or List of covRle objects of equal length and lengths |
| `fraction` | character, default `names(list)`. Names to elements of list, can be integers, as readlengths etc. |

## Value

a covRleList object

## See Also

Other covRLE: `covRle`, `covRle-class`, `covRleFromGR()`, `covRleList-class`

## Examples

```
covRleList(List(covRle()))
```

## covRleList-class *List of covRle*

### Description

Given a run of coverage(x) where x are reads, this covRle combines the 2 strands into 1 object This list can again combine these into 1 object, with accession functions and generalizations.

### Value

a covRleList object

### See Also

Other covRLE: `covRle`, `covRle-class`, `covRleFromGR()`, `covRleList`

## create.experiment *Create an ORFik* `experiment`

### Description

Create a single R object that stores and controls all results relevant to a specific Next generation sequencing experiment. Click the experiment link above in the title if you are not sure what an ORFik experiment is.

By using files in a folder / folders. It will make an experiment table with information per sample, this object allows you to use the extensive API in ORFik that works on experiments.

Information Auto-detection:
There will be several columns you can fill in, when creating the object, if the files have logical names like (RNA-seq_WT_rep1.bam) it will try to auto-detect the most likely values for the columns. Like if it is RNA-seq or Ribo-seq, Wild type or mutant, is this replicate 1 or 2 etc.
You will have to fill in the details that were not auto detected. Easiest way to fill in the blanks are in a csv editor like libre Office or excel. You can also remake the experiment and specify the specific column manually. Remember that each row (sample) must have a unique combination of values. An extra column called "reverse" is made if there are paired data, like +/- strand wig files.

### Usage

```
create.experiment(
  dir,
  exper,
  saveDir = ORFik::config()["exp"],
  txdb = "",
  fa = "",
```

```
  organism = "",
  assembly = "",
  pairedEndBam = FALSE,
  viewTemplate = FALSE,
  types = c("bam", "bed", "wig", "ofst"),
  libtype = "auto",
  stage = "auto",
  rep = "auto",
  condition = "auto",
  fraction = "auto",
  author = "",
  files = findLibrariesInFolder(dir, types, pairedEndBam),
  result_folder = NULL,
  runIDs = extract_run_id(files)
)
```

## Arguments

| | |
|---|---|
| dir | Which directory / directories to create experiment from, must be a directory with NGS data from your experiment. Will include all files of file type specified by "types" argument. So do not mix files from other experiments in the same folder! |
| exper | Short name of experiment. Will be name used to load experiment, and name shown when running `list.experiments` |
| saveDir | Directory to save experiment csv file, default: `ORFik::config()["exp"]`, which has default: "~/Bio_data/ORFik_experiments/". Set to NULL if you don't want to save it to disc. |
| txdb | A path to TxDb (prefered) or gff/gtf (not adviced, slower) file with transcriptome annotation for the organism. |
| fa | A path to fasta genome/sequences used for libraries, remember the file must have a fasta index too. |
| organism | character, default: "" (no organism set), scientific name of organism. Homo sapiens, Danio rerio, Rattus norvegicus etc. If you have a SRA metadata csv file, you can set this argument to study$ScientificName[1], where study is the SRA metadata for all files that was aligned. |
| assembly | character, default: "" (no assembly set). The genome assembly name, like GRCh38 etc. Useful to add if you want detailed metadata of experiment analysis. |
| pairedEndBam | logical FALSE, else TRUE, or a logical list of TRUE/FALSE per library you see will be included (run first without and check what order the files will come in) 1 paired end file, then two single will be c(T, F, F). If you have a SRA metadata csv file, you can set this argument to study$LibraryLayout == "PAIRED", where study is the SRA metadata for all files that was aligned. |
| viewTemplate | run View() on template when finished, default (FALSE). Usually gives you a better view of result than using print(). |
| types | Default `c("bam", "bed", "wig", "ofst")`, which types of libraries to allow as NGS data. |

| | |
|---|---|
| libtype | character, default "auto". Library types, must be length 1 or equal length of number of libraries. "auto" means ORFik will try to guess from file names. Example: RFP (Ribo-seq), RNA (RNA-seq), CAGE, SSU (TCP-seq 40S), LSU (TCP-seq 80S). |
| stage | character, default "auto". Developmental stage, tissue or cell line, must be length 1 or equal length of number of libraries. "auto" means ORFik will try to guess from file names. Example: HEK293 (Cell line), Sphere (zebrafish stage), ovary (Tissue). |
| rep | character, default "auto". Replicate numbering, must be length 1 or equal length of number of libraries. "auto" means ORFik will try to guess from file names. Example: 1 (rep 1), 2 rep(2). Insert only numbers here! |
| condition | character, default "auto". Library conditions, must be length 1 or equal length of number of libraries. "auto" means ORFik will try to guess from file names. Example: WT (wild type), mutant, etc. |
| fraction | character, default "auto". Fractionation of library, must be length 1 or equal length of number of libraries. "auto" means ORFik will try to guess from file names. This columns is used to make experiment unique, if the other columns are not sufficient. Example: cyto (cytosolic fraction), dmso (dmso treated fraction), etc. |
| author | character, default "". Main author of experiment, usually last name is enough. When printing will state "author et al" in info. |
| files | character vector or data.table of library paths in dir. Default: `findLibrariesInFolder(dir, types, pairedEndBam)`. Do not touch unless you want to do some subsetting, it will automatically remove files that are not of file format defined by 'type' argument. Note that sorting on number that: 10 is before 2, so 1, 2, 10, is sorted as: 1, 10, 2. If you want to fix this, you could update this argument with: ORFik:::findLibrariesInFolder()[1,3,2] to get order back to 1,2,10 etc. |
| result_folder | character, default NULL. The folder to output analysis results like QC, count tables etc. By default the libFolder(df) folder is used, the folder of first library in experiment. If you are making a new experiment which is a collection of other experiments, set this to a new folder, to not contaminate your other experiment directories. |
| runIDs | character ids, usually SRR, ERR, or DRR identifiers, default is to search for any of these 3 in the filename by: `extract_run_id(files)`. They are optional. |

## Value

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

## See Also

Other ORFik_experiment: `ORFik.template.experiment()`, `ORFik.template.experiment.zf()`, `bamVarName()`, `experiment-class`, `filepath()`, `libraryTypes()`, `organism,experiment-method`, `outputLibs()`, `read.experiment()`, `save.experiment()`, `validateExperiments()`

## Examples

```
# 1. Pick directory
dir <- system.file("extdata/Homo_sapiens_sample", "", package = "ORFik")
# 2. Pick an experiment name
exper <- "ORFik"
# 3. Pick .gff/.gtf location
txdb <- system.file("extdata/references/homo_sapiens",
                    "Homo_sapiens_dummy.gtf.db", package = "ORFik")
# 4. Pick fasta genome of organism
fa <- system.file("extdata/references/homo_sapiens",
                  "Homo_sapiens_dummy.fasta", package = "ORFik")
# 5. Set organism (optional)
org <- "Homo sapiens"

# Create temple not saved on disc yet:
template <- create.experiment(dir = dir, exper, txdb = txdb,
                              saveDir = NULL,
                              fa = fa, organism = org,
                              viewTemplate = FALSE)
## Now fix non-unique rows: either is libre office, microsoft excel, or in R
template$X5[6] <- "heart"
# read experiment (if you set correctly)
df <- read.experiment(template)
# Save with: save.experiment(df, file = "path/to/save/experiment.csv")

## Create and save experiment directly:
## Default location of experiments is ORFik::config()["exp"]
#template <- create.experiment(dir = dir, exper, txdb = txdb,
#                              fa = fa, organism = org,
#                              viewTemplate = FALSE)
## Custom location (If you work in a team, use a shared folder)
#template <- create.experiment(dir = dir, exper, txdb = txdb,
#                              saveDir = "~/MY/CUSTOME/LOCATION",
#                              fa = fa, organism = org,
#                              viewTemplate = FALSE)
```

---

`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)
```

---

## DEG.analysis *Run differential TE analysis*

---

### Description

Expression analysis of 1 dimension, usually between conditions of RNA-seq.
Using the standardized DESeq2 pipeline flow.
Creates a DESeq model (given x is the target.contrast argument) (usually 'condition' column)
1. RNA-seq model: design = ~ x (differences between the x groups in RNA-seq)

## Usage

```
DEG.analysis(
  df,
  target.contrast = design[1],
  design = ORFik::design(df),
  p.value = 0.05,
  counts = countTable(df, "mrna", type = "summarized"),
  batch.effect = TRUE,
  pairs = combn.pairs(unlist(df[, target.contrast]))
)
```

## Arguments

| | |
|---|---|
| df | an experiment of usually RNA-seq. |
| target.contrast | a character vector, default design[1]. The column in the ORFik experiment that represent the comparison contrasts. By default: the first design factor of the full experimental design. This is the factor you will do the comparison on. DESeq will normalize the counts based on the full design, but the log fold change values will be based on this contrast only. It is usually the 'condition' column. |
| design | a character vector, default design(df.rfp). The full experiment design. Which factors have more than 1 level. Example: stage column are all HEK293, so it can not be a design factor. The condition column has 2 possible values, WT and mutant, so it is a factor of the experiment. Replicates column is not part of design, that is inserted later with setting batch.effect = TRUE. Library type 'libtype' column, can also no be part of initial design, it is always added inside the function, after initial setup. |
| p.value | a numeric, default 0.05 in interval (0,1). Defines adjusted p-value to be used as significance threshold for the result groups. I.e. for exclusive translation group significant subset for p.value = 0.05 means: TE$padj < 0.05 & Ribo$padj < 0.05 & RNA$padj > 0.05. |
| counts | a SummarizedExperiment, default: countTable(df, "mrna", type = "summarized"), all transcripts. Assign a subset if you don't want to analyze all genes. It is recommended to not subset, to give DESeq2 data for variance analysis. |
| batch.effect | logical, default TRUE. Makes replicate column of the experiment part of the design.<br>If you believe you might have batch effects, keep as TRUE. Batch effect usually means that you have a strong variance between biological replicates. Check out pcaExperiment and see if replicates cluster together more than the design factor, to verify if you need to set it to TRUE. |
| pairs | list of character pairs, the experiment contrasts. Default: combn.pairs(unlist(df.rfp[, target.contrast]) |

## Details

#' Analysis is done between each possible combination of levels in the target contrast If target contrast is the condition column, with factor levels: WT, mut1 and mut2 with 3 replicates each. You

get comparison of WT vs mut1, WT vs mut2 and mut1 vs mut2.
The respective result categories are defined as: (given a user defined p value, shown here as 0.05):
Significant - p-value adjusted < 0.05 (p-value cutoff decided by 'p.value argument)

The LFC values are shrunken by lfcShrink(type = "normal").

Remember that DESeq by default can not do global change analysis, it can only find subsets with changes in LFC!

## Value

a data.table with columns: (contrast variable, gene id, regulation status, log fold changes, p.adjust values, mean counts)

## References

doi: 10.1002/cpmb.108

## See Also

Other DifferentialExpression: `DEG.plot.static()`, `DEG_model()`, `DTEG.plot()`, `te.table()`, `te_rna.plot()`

## Examples

```
## Simple example (use ORFik template, then use only RNA-seq)
df <- ORFik.template.experiment()
df.rna <- df[df$libtype == "RNA",]
design(df.rna) # The full experimental design
design(df.rna)[1] # Default target contrast
#dt <- DEG.analysis(df.rna)
```

---

# DEG.plot.static *Plot DEG result*

---

## Description

Plot setup:
X-axis: mean counts Y-axis: Log2 fold changes For explanation of plot, see `DEG.analysis`

## Usage

```
DEG.plot.static(
  dt,
  output.dir = NULL,
  p.value.label = 0.05,
  plot.title = "",
  plot.ext = ".pdf",
```

```
  width = 6,
  height = 6,
  dot.size = 0.4,
  xlim = "auto",
  ylim = "bidir.max",
  relative.name = paste0("DEG_plot", plot.ext)
)
```

## Arguments

| | |
|---|---|
| dt | a data.table with the results from DEG.analysis |
| output.dir | a character path, default NULL(no save), or a directory to save to a file. Relative name of file, specified by 'relative.name' argument. |
| p.value.label | a numeric, default 0.05 in interval (0,1) or "" to not show. What p-value used for the analysis? Will be shown as a caption. |
| plot.title | title for plots, usually name of experiment etc |
| plot.ext | character, default: ".pdf". Alternatives: ".png" or ".jpg". |
| width | numeric, default 6 (in inches) |
| height | numeric, default 6 (in inches) |
| dot.size | numeric, default 0.4, size of point dots in plot. |
| xlim | numeric vector or character preset, default: "bidir.max" (Equal in both + / - direction, using max value + 0.5 of meanCounts column in dt). If you want ggplot to decide limit, set to "auto". For numeric vector, specify min and max x limit: like c(-5, 5) |
| ylim | numeric vector or character preset, default: "bidir.max" (Equal in both + / - direction, using max value + 0.5 of LFC column in dt). If you want ggplot to decide limit, set to "auto". For numeric vector, specify min and max y limit: like c(-10, 10) |
| relative.name | character, Default: paste0("DEG_plot", plot.ext) Relative name of file to be saved in folder specified in output.dir. Change to .pdf if you want pdf file instead of png. |

## Value

a ggplot object

## See Also

Other DifferentialExpression: DEG_model(), DTEG.analysis(), DTEG.plot(), te.table(), te_rna.plot()

## Examples

```
df <- ORFik.template.experiment()
df.rna <- df[df$libtype == "RNA",]
#dt <- DEG.analysis(df.rna)
#Default scaling
#DEG.plot.static(dt)
```

```
#Manual scaling
#DEG.plot.static(dt, xlim = c(-2, 2), ylim = c(-2, 2))
```

---

DEG_model *Get DESeq2 model without running results*

---

## Description

This is the preparation step of DESeq2 analysis using ORFik::DEG.analysis. It is exported so that you can do this step in standalone, usually you want to use DEG.analysis directly.

## Usage

```
DEG_model(
  df,
  target.contrast = design[1],
  design = ORFik::design(df),
  p.value = 0.05,
  counts = countTable(df, "mrna", type = "summarized"),
  batch.effect = TRUE
)
```

## Arguments

| | |
|---|---|
| df | an experiment of usually RNA-seq. |
| target.contrast | a character vector, default design[1]. The column in the ORFik experiment that represent the comparison contrasts. By default: the first design factor of the full experimental design. This is the factor you will do the comparison on. DESeq will normalize the counts based on the full design, but the log fold change values will be based on this contrast only. It is usually the 'condition' column. |
| design | a character vector, default design(df.rfp). The full experiment design. Which factors have more than 1 level. Example: stage column are all HEK293, so it can not be a design factor. The condition column has 2 possible values, WT and mutant, so it is a factor of the experiment. Replicates column is not part of design, that is inserted later with setting batch.effect = TRUE. Library type 'libtype' column, can also no be part of initial design, it is always added inside the function, after initial setup. |
| p.value | a numeric, default 0.05 in interval (0,1). Defines adjusted p-value to be used as significance threshold for the result groups. I.e. for exclusive translation group significant subset for p.value = 0.05 means: TE$padj < 0.05 & Ribo$padj < 0.05 & RNA$padj > 0.05. |
| counts | a SummarizedExperiment, default: countTable(df, "mrna", type = "summarized"), all transcripts. Assign a subset if you don't want to analyze all genes. It is recommended to not subset, to give DESeq2 data for variance analysis. |

batch.effect — logical, default TRUE. Makes replicate column of the experiment part of the design.
If you believe you might have batch effects, keep as TRUE. Batch effect usually means that you have a strong variance between biological replicates. Check out pcaExperiment and see if replicates cluster together more than the design factor, to verify if you need to set it to TRUE.

## Value

a DESeqDataSet object with results stored as metadata columns.

## See Also

Other DifferentialExpression: DEG.plot.static(), DTEG.analysis(), DTEG.plot(), te.table(), te_rna.plot()

## Examples

```
## Simple example (use ORFik template, then use only RNA-seq)
df <- ORFik.template.experiment()
df.rna <- df[df$libtype == "RNA",]
design(df.rna) # The full experimental design
target.contrast <- design(df.rna)[1] # Default target contrast
#ddsMat_rna <- DEG_model(df.rna, target.contrast)
```

---

# DEG_model_results *Get DESeq2 model results from DESeqDataSet*

---

## Description

Get DESeq2 model results from DESeqDataSet

## Usage

```
DEG_model_results(ddsMat_rna, target.contrast, pairs, p.value = 0.05)
```

## Arguments

ddsMat_rna — a DESeqDataSet object with results stored as metadata columns.

target.contrast — a character vector, default design[1]. The column in the ORFik experiment that represent the comparison contrasts. By default: the first design factor of the full experimental design. This is the factor you will do the comparison on. DESeq will normalize the counts based on the full design, but the log fold change values will be based on this contrast only. It is usually the 'condition' column.

pairs — list of character pairs, the experiment contrasts. Default: combn.pairs(unlist(df.rfp[, target.contrast])

p.value a numeric, default 0.05 in interval (0,1). Defines adjusted p-value to be used as significance threshold for the result groups. I.e. for exclusive translation group significant subset for p.value = 0.05 means: TE$padj < 0.05 & Ribo$padj < 0.05 & RNA$padj > 0.05.

## Value

a data.table

## Examples

```
## Simple example (use ORFik template, then use only RNA-seq)
df <- ORFik.template.experiment()
df.rna <- df[df$libtype == "RNA",]
design(df.rna) # The full experimental design
target.contrast <- design(df.rna)[1] # Default target contrast
#ddsMat_rna <- DEG_model(df.rna, target.contrast)
#pairs <- combn.pairs(unlist(df[, target.contrast]))
#dt <- DEG_model_results(ddsMat_rna, target.contrast, pairs)
```

---

# DEG_model_simple *Simple Fpkm ratio test DEG*

---

## Description

If you do not have a valid DESEQ2 experimental setup (contrast), you can use this simplified test

## Usage

```
DEG_model_simple(
  df,
  target.contrast = design[1],
  design = ORFik::design(df),
  p.value = 0.05,
  counts = countTable(df, "mrna", type = "summarized"),
  batch.effect = FALSE
)
```

## Arguments

df an experiment of usually RNA-seq.

target.contrast a character vector, default design[1]. The column in the ORFik experiment that represent the comparison contrasts. By default: the first design factor of the full experimental design. This is the factor you will do the comparison on. DESeq will normalize the counts based on the full design, but the log fold change values will be based on this contrast only. It is usually the 'condition' column.

`design` a character vector, default `design(df.rfp)`. The full experiment design. Which factors have more than 1 level. Example: stage column are all HEK293, so it can not be a design factor. The condition column has 2 possible values, WT and mutant, so it is a factor of the experiment. Replicates column is not part of design, that is inserted later with setting `batch.effect = TRUE`. Library type 'libtype' column, can also no be part of initial design, it is always added inside the function, after initial setup.

`p.value` a numeric, default 0.05 in interval (0,1). Defines adjusted p-value to be used as significance threshold for the result groups. I.e. for exclusive translation group significant subset for p.value = 0.05 means: TE$padj < 0.05 & Ribo$padj < 0.05 & RNA$padj > 0.05.

`counts` a SummarizedExperiment, default: countTable(df, "mrna", type = "summarized"), all transcripts. Assign a subset if you don't want to analyze all genes. It is recommended to not subset, to give DESeq2 data for variance analysis.

`batch.effect` logical, default TRUE. Makes replicate column of the experiment part of the design.
If you believe you might have batch effects, keep as TRUE. Batch effect usually means that you have a strong variance between biological replicates. Check out `pcaExperiment` and see if replicates cluster together more than the design factor, to verify if you need to set it to TRUE.

## Value

a data.table of fpkm ratios

## Examples

```
## Simple example (use ORFik template, then use only RNA-seq)
df <- ORFik.template.experiment()
df <- df[df$libtype == "RNA",]
#dt <- DEG_model_simple(df)
```

---

`design,experiment-method`

*Get experimental design Find the column/columns that create a separation between samples, by default skips replicate and choose first that is from either: libtype, condition, stage and fraction.*

---

## Description

Get experimental design Find the column/columns that create a separation between samples, by default skips replicate and choose first that is from either: libtype, condition, stage and fraction.

### Usage

```
## S4 method for signature 'experiment'
design(
  object,
  batch.correction.design = FALSE,
  as.formula = FALSE,
  multi.factor = TRUE
)
```

### Arguments

| | |
|---|---|
| object | an ORFik experiment |
| batch.correction.design | logical, default FALSE. If true, add replicate as a second design factor (only if >= 2 replicates exists). |
| as.formula | logical, default FALSE. If TRUE, return as formula |
| multi.factor | logical, default TRUE If FALSE, return first factor only (+ rep, if batch.correction.design is true). Order of picking is: libtype, if not then: stage, if not then: condition, if not then: fraction. |

### Value

a character (name of column) or a formula

### Examples

```
df <- ORFik.template.experiment()
design(df) # The 2 columns that decides the design here
# If we subset it changes
design(df[df$libtype == "RFP",])
# Only single factor design, it picks first
design(df, multi.factor = FALSE)
```

---

## 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. The E-site will be shift + 3 and A site will be shift - 3. So update to these, if you rather want those.

## Usage

```
detectRibosomeShifts(
  footprints,
  txdb,
  start = TRUE,
  stop = FALSE,
  top_tx = 10L,
  minFiveUTR = 30L,
  minCDS = 150L,
  minThreeUTR = if (stop) {
     30
 } else NULL,
  txNames = filterTranscripts(txdb, minFiveUTR, minCDS, minThreeUTR),
  firstN = 150L,
  tx = NULL,
  min_reads = 1000,
  min_reads_TIS = 50,
  accepted.lengths = 26:34,
  heatmap = FALSE,
  must.be.periodic = TRUE,
  strict.fft = TRUE,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| footprints | GAlignments object of RiboSeq reads - footprints, can also be path to the .bam /.ofst file. If GAlignment object has a meta column called "score", this will be used as replicate numbering for that read. So be careful if you have custom files with score columns, with another meaning. |
| txdb | a TxDb file, a path to one of: (.gtf ,.gff, .gff2, .gff2, .db or .sqlite) or an ORFik experiment |
| start | (logical) Whether to include predictions based on the start codons. Default TRUE. |
| stop | (logical) Whether to include predictions based on the stop codons. Default FASLE. Only use if there exists 3' UTRs for the annotation. If peridicity around stop codon is stronger than at the start codon, use stop instead of start region for p-shifting. |
| top_tx | (integer), default 10. Specify which % of the top TIS coverage transcripts to use for estimation of the shifts. By default we take top 10 top covered transcripts as they represent less noisy data-set. This is only applicable when there are more than 1000 transcripts. |
| minFiveUTR | (integer) minimum bp for 5' UTR during filtering for the transcripts. Set to NULL if no 5' UTRs exists for annotation. |
| minCDS | (integer) minimum bp for CDS during filtering for the transcripts |
| minThreeUTR | (integer) minimum bp for 3' UTR during filtering for the transcripts. Set to NULL if no 3' UTRs exists for annotation. |

`txNames` a character vector of subset of CDS to use. Default: txNames = filterTranscripts(txdb, minFiveUTR, minCDS, minThreeUTR)
Example: c("ENST1000005"), will use only that transcript (You should use at least 100!). Remember that top_tx argument, will by default specify to use top 10 % of those CDSs. Set that to 100, to use all these specified transcripts.

`firstN` (integer) Represents how many bases of the transcripts downstream of start codons to use for initial estimation of the periodicity.

`tx` a GRangesList, if you do not have 5' UTRs in annotation, send your own version. Example: extendLeaders(tx, 30) Where 30 bases will be new "leaders". Since each original transcript was either only CDS or non-coding (filtered out).

`min_reads` default (1000), how many reads must a read-length have in total to be considered for periodicity.

`min_reads_TIS` default (50), how many reads must a read-length have in the TIS region to be considered for periodicity.

`accepted.lengths` accepted read lengths, default 26:34, usually ribo-seq is strongest between 27:32.

`heatmap` a logical or character string, default FALSE. If TRUE, will plot heatmap of raw reads before p-shifting to console, to see if shifts given make sense. You can also set a filepath to save the file there.

`must.be.periodic` logical TRUE, if FALSE will not filter on periodic read lengths. (The Fourier transform filter will be skipped). This is useful if you are not going to do periodicity analysis, that is: for you more coverage depth (more read lengths) is more important than only keeping the high quality periodic read lengths.

`strict.fft` logical, TRUE. Use a FFT without noise filter. This means keep only reads lengths that are "periodic for the human eye". If you want more coverage, set to FALSE, to also get read lengths that are "messy", but the noise filter detects the periodicity of 3. This should only be done when you do not need high quality periodic reads! Example would be differential translation analysis by counts over each ORF.

`verbose` logical, default FALSE. Report details of analysis/periodogram. Good if you are not sure if the analysis was correct.

## 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.

For how the Fourier transform works, see: `isPeriodic`
For how the changepoint analysis works, see: `changePointAnalysis`

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

## Value

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

## References

https://bmcgenomics.biomedcentral.com/articles/10.1186/s12864-018-4912-6

## See Also

Other pshifting: changePointAnalysis(), shiftFootprints(), shiftFootprintsByExperiment(), shiftPlots(), shifts_load(), shifts_save()

## Examples

```
## Basic run
# Transcriptome annotation ->
gtf_file <- system.file("extdata/references/danio_rerio", "annotations.gtf", package = "ORFik")
# Ribo seq data ->
riboSeq_file <- system.file("extdata/Danio_rerio_sample", "ribo-seq.bam", package = "ORFik")
## Not run:
footprints <- readBam(riboSeq_file)
## Using CDS start site as reference point:
detectRibosomeShifts(footprints, gtf_file)
## Using CDS start site and stop site as 2 reference points:
#detectRibosomeShifts(footprints, gtf_file, stop = TRUE)
## Debug and detailed information for accepted reads lengths and p-site:
detectRibosomeShifts(footprints, gtf_file, heatmap = TRUE, verbose = TRUE)
## Debug why read length 31 was not accepted or wrong p-site:
#detectRibosomeShifts(footprints, gtf_file, must.be.periodic = FALSE,
#              accepted.lengths = 31, heatmap = TRUE, verbose = TRUE)

## Subset bam file
param = ScanBamParam(flag = scanBamFlag(
                       isDuplicate = FALSE,
                       isSecondaryAlignment = FALSE))
footprints <- readBam(riboSeq_file, param = param)
detectRibosomeShifts(footprints, gtf_file, stop = TRUE)

## Without 5' Annotation
library(GenomicFeatures)

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


## End(Not run)
```

`detect_ribo_orfs` *Detect ORFs by Ribosome profiling data*

## Description

Finding all ORFs: 1. Find all ORFs in mRNA using ORFik findORFs, with defined parameters.
To create the candidate ORFs (all ORFs returned):
Steps (candidate set):
Define a candidate search set by these 3 rules:
1.a Allowed ORF type: uORF, NTE, etc (only keep these in candidate list)
1.b Must have at least x reads over whole orf (default 10 reads)
1.c Must have at least x reads over start site (default 3 reads)
The total list is defined by these names, and saved according to allowed ORF type/types.
To create the prediction status (TRUE/FALSE) per candidate
Steps (prediction status)
(UP_NT is a 20nt window upstream of ORF, that stops 2NT before ORF starts) :
1. ORF mean reads per NT > (UP_NT mean reads per NT * 1.3)
2. ORFScore > 2.5
3. TIS total reads + 3 > ORF median reads per NT
4. Given expression above, a TRUE prediction is defined with the AND operatior: 1. & 2. & 3.

In code that is:
`predicted <- (orfs_cov_stats$mean > upstream_cov_stats$mean*1.3) & orfs_cov_stats$ORFScores > 2.5 & ((reads_start[candidates] + 3) > orfs_cov_stats$median)`

## Usage

```
detect_ribo_orfs(
  df,
  out_folder,
  ORF_categories_to_keep,
 prefix_result = paste(c(ORF_categories_to_keep, gsub(" ", "_", organism(df))), collapse
    = "_"),
  mrna = loadRegion(df, "mrna"),
  cds = loadRegion(df, "cds"),
  libraries = outputLibs(df, type = "pshifted", output = "envirlist"),
 orf_candidate_ranges = findORFs(seqs = txSeqsFromFa(mrna, df, TRUE), longestORF =
    longestORF, startCodon = startCodon, stopCodon = stopCodon, minimumLength =
    minimumLength),
  export_metrics_table = TRUE,
  longestORF = FALSE,
  startCodon = startDefinition(1),
  stopCodon = stopDefinition(1),
  minimumLength = 0,
  minimum_reads_ORF = 10,
  minimum_reads_start = 3
)
```

## Arguments

| | |
|---|---|
| df | an ORFik experiment |
| out_folder | Directory to save files |
| ORF_categories_to_keep | options, any subset of: c("uORF", "uoORF", "annotated", "NTE","NTT", "internal", "doORF", "dORF", "a_error").<br>• uORF: Upstream ORFs (Starting in 5' UTR), not overlapping CDS<br>• uoORF: Upstream ORFs (Starting in 5' UTR), overlapping CDS<br>• annotated: The defined CDS for that transcript<br>• NTE: 5' Start codon extension of annotated CDS<br>• NTT: 5' Start codon truncation of annotated CDS<br>• internal: Starting inside CDS, ending before CDS ends<br>• doORF: Downstream ORFs (Ending in 3' UTR), overlapping CDS<br>• dORF: Downstream ORFs (Ending in 3' UTR), not overlapping CDS<br>• a_error: Any ORF detect not in the above categories |
| prefix_result | the prefix name of output files to out_folder. Default: paste(c(ORF_categories_to_keep, gsub(" ", "_", organism(df))), collapse = "_") |
| mrna | = loadRegion(df, "mrna") |
| cds | = loadRegion(df, "cds") |
| libraries | the ribo-seq libraries loaded into R as list, default: outputLibs(df, type = "pshifted", output = "envirlist") |
| orf_candidate_ranges | IRangesList, = findORFs(seqs = txSeqsFromFa(mrna, df, TRUE),longestORF = longestORF, startCodon = startCodon, stopCodon = stopCodon,minimumLength = minimumLength) |
| export_metrics_table | logical, default TRUE. Export table of statistics to file with suffix: "_prediction_table.rds" |
| longestORF | (logical) Default TRUE. Keep only the longest ORF per unique stopcodon: (seqname, strand, stopcodon) combination, Note: Not longest per transcript! You can also use function longestORFs after creation of ORFs for same result. |
| startCodon | (character vector) Possible START codons to search for. Check startDefinition for helper function. Note that it is case sensitive, so "atg" would give 0 hits for a sequence with only capital "ATG" ORFs. |
| stopCodon | (character vector) Possible STOP codons to search for. Check stopDefinition for helper function. Note that it is case sensitive, so "tga" would give 0 hits for a sequence with only capital "TGA" ORFs. |
| minimumLength | (integer) Default is 0. Which is START + STOP = 6 bp. Minimum length of ORF, without counting 3bps 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. |
| minimum_reads_ORF | numeric, default 10, orf removed if less reads overlap whole orf |
| minimum_reads_start | numeric, default 3, orf removed if less reads overlap start |

## Value

invisible(NULL), all ORF results saved to disc

## Examples

```
# Pre requisites
# 1. Create ORFik experiment
#  ORFik::create.experiment(...)
# 2. Create ORFik optimized annotation:
# makeTxdbFromGenome(gtf = ORFik:::getGtfPathFromTxdb(df), genome = df@fafile,
#                       organism = organism(df), optimize = TRUE)
# 3. There must exist pshifted reads, either as default files, or in a relative folder called
# "./pshifted/". See ?shiftFootprintsByExperiment
# EXAMPLE:
df <- ORFik.template.experiment()
df <- df[df$libtype == "RFP",][c(1,2),]
result_folder <- riboORFsFolder(df, tempdir())
results <- detect_ribo_orfs(df, result_folder, c("uORF", "uoORF", "annotated", "NTE"))

# Load results of annotated ORFs
table <- riboORFs(df[1,], type = "table", result_folder)
table # See all statistics
sum(table$predicted) # How many were predicted as Ribo-seq ORFs
# Load 2 results
table <- riboORFs(df[1:2,], type = "table", result_folder)
table # See all statistics
sum(table$predicted) # How many were predicted as Ribo-seq ORFs

# Load GRangesList
candidates_gr <- riboORFs(df[1,], type = "ranges_candidates", result_folder)
prediction <- riboORFs(df[1,], type = "predictions", result_folder)

predicted_gr <- riboORFs(df[1:2,], type = "ranges_predictions", result_folder)
identical(predicted_gr[[1]], candidates_gr[[1]][prediction[[1]]])
## Inspect predictions in RiboCrypt
# library(RiboCrypt)
# Inspect Predicted
view <- predicted_gr[[1]][1]
#multiOmicsPlot_ORFikExp(view, df, view, leader_extension = 100, trailer_extension = 100)
# Inspect not predicted
view <- candidates_gr[[1]][!prediction[[1]]][1]
#multiOmicsPlot_ORFikExp(view, df, view, leader_extension = 100, trailer_extension = 100)
```

---

# disengagementScore *Disengagement score (DS)*

---

## Description

Disengagement score is defined as

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

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

## Usage

```
disengagementScore(
  grl,
  RFP,
  GtfOrTx,
  RFP.sorted = FALSE,
  weight = 1L,
  overlapGrl = NULL
)
```

## Arguments

| | |
|---|---|
| grl | a GRangesList object with usually either leaders, cds', 3' utrs or ORFs. |
| RFP | RiboSeq reads as GAlignments, 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 (FALSE), an optimizer, have you ran this line: RFP <- sort(RFP[countOverlaps(RFP, tx, type = "within") > 0]) Normally not touched, for internal optimization purposes. |
| weight | a vector (default: 1L, if 1L it is identical to countOverlaps()), if single number (!= 1), it applies for all, if more than one must be equal size of 'reads'. else it must be the string name of a defined meta column in subject "reads", that gives number of times a read was found. GRanges("chr1", 1, "+", score = 5), would mean "score" column tells that this alignment region was found 5 times. |
| overlapGrl | an integer, (default: NULL), if defined must be countOverlaps(grl, RFP), added for speed if you already have it |

## Value

a named vector of numeric values of scores

## References

doi: 10.1242/dev.098344

## See Also

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

## Examples

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

---

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

---

## Description

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

## Usage

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

## Arguments

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

## Value

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

## References

doi: 10.1074/jbc.R116.733899

## See Also

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

## Examples

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

---

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

---

## Description

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

## Usage

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

## Arguments

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

## Value

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

## References

doi: 10.1074/jbc.R116.733899

## See Also

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

## Examples

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

download.SRA *Download read libraries from SRA*

## Description

Multicore version download, see documentation for SRA toolkit for more information.

## Usage

```
download.SRA(
  info,
  outdir,
  rename = TRUE,
  fastq.dump.path = install.sratoolkit(),
  settings = paste("--skip-technical", "--split-files"),
  subset = NULL,
  compress = TRUE,
  use.ebi.ftp = is.null(subset),
  ebiDLMethod = "auto",
  timeout = 5000,
  BPPARAM = bpparam()
)
```

## Arguments

| | |
|---|---|
| info | character vector of only SRR numbers or a data.frame with SRA metadata information including the SRR numbers in a column called "Run" or "SRR". Can be SRR, ERR or DRR numbers. If only SRR numbers can not rename, since no additional information is given. |
| outdir | directory to store runs, files are named by default (rename = TRUE) by information from SRA metadata table, if (rename = FALSE) named according to SRR numbers. |
| rename | logical or character, default TRUE (Auto guess new names). False: Skip renaming. A character vector of equal size as files wanted can also be given. Priority of renaming from the metadata is to check for unique names in the LibraryName column, then the sample_title column if no valid names in LibraryName. If new names found and still duplicates, will add "_rep1", "_rep2" to make them unique. If no valid names, will not rename, that is keep the SRR numbers, you then can manually rename files to something more meaningful. |
| fastq.dump.path | path to fastq-dump binary, default: path returned from install.sratoolkit() |
| settings | a string of arguments for fastq-dump, default: paste("–gzip", "–skip-technical", "–split-files") |
| subset | an integer or NULL, default NULL (no subset). If defined as a integer will download only the first n reads specified by subset. If subset is defined, will force to use fastq-dump which is slower than ebi download. |

| | |
|---|---|
| compress | logical, default TRUE. Download compressed files ".gz". |
| use.ebi.ftp | logical, default: is.null(subset). Use ORFiks much faster download function that only works when subset is null, if subset is defined, it uses fastqdump, it is slower but supports subsetting. Force it to use fastqdump by setting this to FALSE. |
| ebiDLMethod | character, default "auto". Which download protocol to use in download.file when using ebi ftp download. Sometimes "curl" is might not work (the default auto usually), in those cases use wget. See "method" argument of ?download.file, for more info. |
| timeout | 5000, how many seconds before killing download if still active? Will overwrite global option until R session is closed. Increase value if you are on a very slow connection or downloading a large dataset. |
| BPPARAM | how many cores/threads to use? default: bpparam(). To see number of threads used, do `bpparam()$workers` |

## Value

a character vector of download files filepaths

## References

https://ncbi.github.io/sra-tools/fastq-dump.html

## See Also

Other sra: `browseSRA()`, `download.SRA.metadata()`, `download.ebi()`, `get_bioproject_candidates()`, `install.sratoolkit()`, `rename.SRA.files()`

## Examples

```
SRR <- c("SRR453566") # Can be more than one

## Simple single SRR run of YEAST
outdir <- tempdir() # Specify output directory
# Download, get 5 first reads
#download.SRA(SRR, outdir, rename = FALSE, subset = 5)

## Using metadata column to get SRR numbers and to be able to rename samples
outdir <- tempdir() # Specify output directory
info <- download.SRA.metadata("SRP226389", outdir) # By study id
## Download, 5 first reads of each library and rename
#files <- download.SRA(info, outdir, subset = 5)
#Biostrings::readDNAStringSet(files[1], format = "fastq")

## Download full libraries of experiment
## (note, this will take some time to download!)
#download.SRA(info, outdir)
```

`download.SRA.metadata` *Downloads metadata from SRA*

## Description

Given a experiment identifier, query information from different locations of SRA to get a complete metadata table of the experiment. It first finds Runinfo for each library, then sample info, if pubmed id is not found searches for that and searches for author through pubmed.

## Usage

```
download.SRA.metadata(
  SRP,
  outdir = tempdir(),
  remove.invalid = TRUE,
  auto.detect = FALSE,
  abstract = "printsave",
  force = FALSE,
  rich.format = FALSE
)
```

## Arguments

| | |
|---|---|
| `SRP` | a string, a study ID as either the PRJ, SRP, ERP, DRPor GSE of the study, examples would be "SRP226389" or "ERP116106". If GSE it will try to convert to the SRP to find the files. The call works as long the runs are registered on the efetch server, as their is a linked SRP link from bioproject or GSE. Example which fails is "PRJNA449388", which does not have a linking like this. |
| `outdir` | directory to save file, default: tempdir(). The file will be called "SraRunInfo_SRP.csv", where SRP is the SRP argument. We advice to use bioproject IDs "PRJNA...". The directory will be created if not existing. |
| `remove.invalid` | logical, default TRUE. Remove Runs with 0 reads (spots) |
| `auto.detect` | logical, default FALSE. If TRUE, ORFik will add additional columns:<br>LIBRARYTYPE: (is this Ribo-seq or mRNA-seq, CAGE etc),<br>REPLICATE: (is this replicate 1, 2 etc),<br>STAGE: (Which time point, cell line or tissue is this, HEK293, TCP-1, 24hpf etc),<br>CONDITION: (is this Wild type control or a mutant etc).<br>These values are only qualified guesses from the metadata, so always double check! |
| `abstract` | character, default "printsave". If abstract for project exists, print and save it (save the file to same directory as runinfo). Alternatives: "print", Only print first time downloaded, will not be able to print later.<br>save" save it, no print<br>"no" skip download of abstract |

| | |
|---|---|
| force | logical, default FALSE. If TRUE, will redownload all files needed even though they exists. Useful if you wanted auto.detection, but already downloaded without it. |
| rich.format | logical, default FALSE. If TRUE, will fetch all Experiment and Sample attributes. It means, that different studies can have different set of columns if set to TRUE. |

## Details

A common problem is that the project is not linked to an article, you will then not get a pubmed id.

The algorithm works like this:
If GEO identifier, find the SRP.
Then search Entrez for project and get sample identifier.
From that extract the run information and collect into a final table.

## Value

a data.table of the metadata, 1 row per sample, SRR run number defined in 'Run' column.

## References

doi: 10.1093/nar/gkq1019

## See Also

Other sra: browseSRA(), download.SRA(), download.ebi(), get_bioproject_candidates(), install.sratoolkit(), rename.SRA.files()

## Examples

```
## Originally on SRA
download.SRA.metadata("SRP226389")
## Now try with auto detection (guessing additional library info)
## Need to specify output dir as tempfile() to re-download
#download.SRA.metadata("SRP226389", tempfile(), auto.detect = TRUE)
## Originally on ENA (RCP-seq data)
# download.SRA.metadata("ERP116106")
## Originally on GEO (GSE) (save to directory to keep info with fastq files)
# download.SRA.metadata("GSE61011")
## Bioproject ID
# download.SRA.metadata("PRJNA231536")
```

DTEG.analysis *Run differential TE analysis*

## Description

Expression analysis of 2 dimensions, usually Ribo-seq vs RNA-seq.
Using an equal reimplementation of the deltaTE algorithm (see reference).
Creates a total of 3 DESeq models (given x is the target.contrast argument) (usually 'condition' column) and libraryType is RNA-seq and Ribo-seq):
1. Ribo-seq model: design = ~ x (differences between the x groups in Ribo-seq)
2. RNA-seq model: design = ~ x (differences between the x groups in RNA-seq)
3. TE model: design = ~ x + libraryType + libraryType:x (differences between the x and libraryType groups and the interaction between them)
You need at least 2 groups and 2 replicates per group. By default, the Ribo-seq counts will be over CDS and RNA-seq counts over whole mRNAs, per transcript.

## Usage

```
DTEG.analysis(
  df.rfp,
  df.rna,
  output.dir = QCfolder(df.rfp),
  target.contrast = design[1],
  design = ORFik::design(df.rfp),
  p.value = 0.05,
  RFP_counts = countTable(df.rfp, "cds", type = "summarized"),
  RNA_counts = countTable(df.rna, "mrna", type = "summarized"),
  batch.effect = FALSE,
  pairs = combn.pairs(unlist(df.rfp[, design])),
  plot.title = "",
  plot.ext = ".pdf",
  width = 6,
  height = 6,
  dot.size = 0.4,
  relative.name = paste0("DTEG_plot", plot.ext),
  complex.categories = FALSE
)
```

## Arguments

| | |
|---|---|
| df.rfp | a experiment of usually Ribo-seq or 80S from TCP-seq. (the numerator of the experiment, usually having a primary role) |
| df.rna | a experiment of usually RNA-seq. (the denominator of the experiment, usually having a normalizing function) |
| output.dir | character, default QCfolder(df.rfp). output.dir directory to save plots, plot will be named "TE_between". If NULL, will not save. |

target.contrast
: a character vector, default design[1]. The column in the ORFik experiment that represent the comparison contrasts. By default: the first design factor of the full experimental design. This is the factor you will do the comparison on. DESeq will normalize the counts based on the full design, but the log fold change values will be based on this contrast only. It is usually the 'condition' column.

design
: a character vector, default design(df.rfp). The full experiment design. Which factors have more than 1 level. Example: stage column are all HEK293, so it can not be a design factor. The condition column has 2 possible values, WT and mutant, so it is a factor of the experiment. Replicates column is not part of design, that is inserted later with setting batch.effect = TRUE. Library type 'libtype' column, can also no be part of initial design, it is always added inside the function, after initial setup.

p.value
: a numeric, default 0.05 in interval (0,1). Defines adjusted p-value to be used as significance threshold for the result groups. I.e. for exclusive translation group significant subset for p.value = 0.05 means: TE$padj < 0.05 & Ribo$padj < 0.05 & RNA$padj > 0.05.

RFP_counts
: a SummarizedExperiment, default: countTable(df.rfp, "cds", type = "summarized"), unshifted libraries, all transcript CDSs. If you have pshifted reads and countTables, do: countTable(df.rfp, "cds", type = "summarized", count.folder = "pshifted") Assign a subset if you don't want to analyze all genes. It is recommended to not subset, to give DESeq2 data for variance analysis.

RNA_counts
: a SummarizedExperiment, default: countTable(df.rna, "mrna", type = "summarized"), all transcripts. Assign a subset if you don't want to analyze all genes. It is recommended to not subset, to give DESeq2 data for variance analysis.

batch.effect
: logical, default TRUE. Makes replicate column of the experiment part of the design.
If you believe you might have batch effects, keep as TRUE. Batch effect usually means that you have a strong variance between biological replicates. Check out pcaExperiment and see if replicates cluster together more than the design factor, to verify if you need to set it to TRUE.

pairs
: list of character pairs, the experiment contrasts. Default: combn.pairs(unlist(df.rfp[, target.contrast])

plot.title
: title for plots, usually name of experiment etc

plot.ext
: character, default: ".pdf". Alternatives: ".png" or ".jpg".

width
: numeric, default 6 (in inches)

height
: numeric, default 6 (in inches)

dot.size
: numeric, default 0.4, size of point dots in plot.

relative.name
: character, Default: paste0("DTEG_plot", plot.ext) Relative name of file to be saved in folder specified in output.dir. Change to .pdf if you want pdf file instead of png.

complex.categories
: logical, default FALSE. Seperate into more groups, will add Inverse (opposite diagonal of mRNA abundance) and Expression (only significant mRNA-seq)

## Details

Log fold changes and p-values are created from a Walds test on the comparison contrast described bellow. The RNA-seq and Ribo-seq LFC values are shrunken using DESeq2::lfcShrink(type = "normal"). Note that the TE LFC values are not shrunken (as following specifications from deltaTE paper)

Analysis is done between each possible combination of levels in the target contrast If target contrast is condition column, with factor levels: WT, mut1 and mut2 with 3 replicates each. You get comparison of WT vs mut1, WT vs mut2 and mut1 vs mut2.
The respective result categories are defined as: (given a user defined p value, shown here as 0.05):
1. Translation - te.p.adj < 0.05 & rfp.p.adj < 0.05 & rna.p.adj > 0.05
2. mRNA abundance - te.p.adj > 0.05 & rfp.p.adj < 0.05 & rna.p.adj > 0.05
3. Buffering - te.p.adj < 0.05 & rfp.p.adj > 0.05 & rna.p.adj > 0.05

Buffering will be broken down into sub-categories if you set complex.categories = TRUE See Figure 1 in the reference article for a clear definition of the groups!
If you do not need isoform variants, subset to longest isoform per gene either before or in the returned object (See examples). If you do not have RNA-seq controls, you can still use DESeq on Ribo-seq alone.
The LFC values are shrunken by lfcShrink(type = "normal").

Remember that DESeq by default can not do global change analysis, it can only find subsets with changes in LFC!

## Value

a data.table with columns: (contrast variable, gene id, regulation status, log fold changes, p.adjust values, mean counts)

## References

doi: 10.1002/cpmb.108

## See Also

Other DifferentialExpression: `DEG.plot.static()`, `DEG_model()`, `DTEG.plot()`, `te.table()`, `te_rna.plot()`

## Examples

```
## Simple example (use ORFik template, then split on Ribo and RNA)
df <- ORFik.template.experiment()
df.rfp <- df[df$libtype == "RFP",]
df.rna <- df[df$libtype == "RNA",]
design(df.rfp) # The experimental design, per libtype
design(df.rfp)[1] # Default target contrast
#dt <- DTEG.analysis(df.rfp, df.rna)
```

```
## If you want to use the pshifted libs for analysis:
#dt <- DTEG.analysis(df.rfp, df.rna,
#                    RFP_counts = countTable(df.rfp, region = "cds",
#                       type = "summarized", count.folder = "pshifted"))
## Restrict DTEGs by log fold change (LFC):
## subset to abs(LFC) < 1.5 for both rfp and rna
#dt[abs(rfp) < 1.5 & abs(rna) < 1.5, Regulation := "No change"]

## Only longest isoform per gene:
#tx_longest <- filterTranscripts(df.rfp, 0, 1, 0)
#dt <- dt[id %in% tx_longest,]
## Convert to gene id
#dt[, id := txNamesToGeneNames(id, df.rfp)]
## To get by gene symbol, use biomaRt conversion
## To flip directionality of contrast pair nr 2:
#design <- "condition"
#pairs <- combn.pairs(unlist(df.rfp[, design])
#pairs[[2]] <- rev(pars[[2]])
#dt <- DTEG.analysis(df.rfp, df.rna,
#                    RFP_counts = countTable(df.rfp, region = "cds",
#                       type = "summarized", count.folder = "pshifted"),
#                       pairs = pairs)
```

---

`DTEG.plot` *Plot DTEG result*

---

## Description

For explanation of plot catagories, see `DTEG.analysis`

## Usage

```
DTEG.plot(
  dt,
  output.dir = NULL,
  p.value.label = 0.05,
  plot.title = "",
  plot.ext = ".pdf",
  width = 6,
  height = 6,
  dot.size = 0.4,
  xlim = "bidir.max",
  ylim = "bidir.max",
  relative.name = paste0("DTEG_plot", plot.ext)
)
```

## Arguments

| | |
|---|---|
| dt | a data.table with the results from DTEG.analysis |
| output.dir | a character path, default NULL(no save), or a directory to save to a file. Relative name of file, specified by 'relative.name' argument. |
| p.value.label | a numeric, default 0.05 in interval (0,1) or "" to not show. What p-value used for the analysis? Will be shown as a caption. |
| plot.title | title for plots, usually name of experiment etc |
| plot.ext | character, default: ".pdf". Alternatives: ".png" or ".jpg". |
| width | numeric, default 6 (in inches) |
| height | numeric, default 6 (in inches) |
| dot.size | numeric, default 0.4, size of point dots in plot. |
| xlim | numeric vector or character preset, default: "bidir.max" (Equal in both + / - direction, using max value + 0.5 of rna column in dt). If you want ggplot to decide limit, set to "auto". For numeric vector, specify min and max x limit: like c(-5, 5) |
| ylim | numeric vector or character preset, default: "bidir.max" (Equal in both + / - direction, using max value + 0.5 of rfp column in dt). If you want ggplot to decide limit, set to "auto". For numeric vector, specify min and max y limit: like c(-10, 10) |
| relative.name | character, Default: paste0("DTEG_plot", plot.ext) Relative name of file to be saved in folder specified in output.dir. Change to .pdf if you want pdf file instead of png. |

## Value

a ggplot object

## See Also

Other DifferentialExpression: DEG.plot.static(), DEG_model(), DTEG.analysis(), te.table(), te_rna.plot()

## Examples

```
df <- ORFik.template.experiment()
df.rfp <- df[df$libtype == "RFP",]
df.rna <- df[df$libtype == "RNA",]
#dt <- DTEG.analysis(df.rfp, df.rna)
#Default scaling
#DTEG.plot(dt)
#Manual scaling
#DTEG.plot(dt, xlim = c(-2, 2), ylim = c(-2, 2))
```

`entropy` *Percentage of maximum entropy*

## Description

Calculates percentage of maximum entropy of the ‘reads‘ coverage over each ORF in ‘grl‘ group. The entropy value per group is a real number in the interval (0:1), where 0 indicates no variance in reads over all codons of group For example c(0,0,0,0) has 0 entropy, since no reads overlap.
Interval: [0]: No reads or all reads in 1 place
Interval: [0.01-0.99]: >= 2 positions covered
Interval: [1]: all positions covered perfectly in frame

## Usage

```
entropy(grl, reads, weight = 1L, is.sorted = FALSE, overlapGrl = NULL)
```

## Arguments

| | |
|---|---|
| `grl` | a `GRangesList` object can be either transcripts, 5’ utrs, cds’, 3’ utrs or ORFs as a special case (uORFs, potential new cds’ etc). If regions are not spliced you can send a `GRanges` object. |
| `reads` | a `GAlignments`, `GRanges` or `GRangesList` object, usually of RiboSeq, RnaSeq, CageSeq, etc. |
| `weight` | a vector (default: 1L, if 1L it is identical to countOverlaps()), if single number (!= 1), it applies for all, if more than one must be equal size of ’reads’. else it must be the string name of a defined meta column in subject "reads", that gives number of times a read was found. GRanges("chr1", 1, "+", score = 5), would mean "score" column tells that this alignment region was found 5 times. |
| `is.sorted` | logical (FALSE), is grl sorted. That is + strand groups in increasing ranges (1,2,3), and - strand groups in decreasing ranges (3,2,1) |
| `overlapGrl` | an integer, (default: NULL), if defined must be countOverlaps(grl, RFP), added for speed if you already have it |

## Value

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

## See Also

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

## Examples

```
# a toy example with ribo-seq p-shifted reads
ORF <- GRangesList(tx1 = GRanges("1", IRanges(1, width = 9), "+"))
entropy(ORF, GRanges()) # 0
entropy(ORF, GRanges("1", IRanges(c(1)), "+")) # 0
entropy(ORF, GRanges("1", IRanges(c(1,4,6,7)), "+")) # 0.94
entropy(ORF, GRanges("1", IRanges(c(1,4,7)), "+", score = c(1,2,1)),
        weight = "score") # 0.94
entropy(ORF, GRanges("1", IRanges(c(1,4,7)), "+")) # Perfect = 1
```

---

`envExp` *Get ORFik experiment environment*

---

## Description

More correctly, get the pointer reference, default is .GlobalEnv

## Usage

```
envExp(x)
```

## Arguments

| | |
|---|---|
| x | an ORFik experiment |

## Value

environment pointer, name of environment: pointer

---

`envExp,experiment-method` *Get ORFik experiment environment*

---

## Description

More correctly, get the pointer reference, default is .GlobalEnv

## Usage

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

## Arguments

| | |
|---|---|
| x | an ORFik experiment |

## Value

environment pointer, name of environment: pointer

## envExp<- *Set ORFik experiment environment*

### Description

More correctly, set the pointer reference, default is .GlobalEnv

### Usage

```
envExp(x) <- value
```

### Arguments

| | |
|---|---|
| x | an ORFik experiment |
| value | environment pointer to assign to experiment |

### Value

an ORFik experiment with updated environment

## envExp<-,experiment-method *Set ORFik experiment environment*

### Description

More correctly, set the pointer reference, default is .GlobalEnv

### Usage

```
## S4 replacement method for signature 'experiment'
envExp(x) <- value
```

### Arguments

| | |
|---|---|
| x | an ORFik experiment |
| value | environment pointer to assign to experiment |

### Value

an ORFik experiment with updated environment

`experiment-class` *experiment class definition*

## Description

It is an object that simplify and error correct your NGS workflow, creating a single R object that stores and controls all results relevant to a specific experiment.
It contains following important parts:

- filepaths : and info for each library in the experiment (for multiple files formats: bam, bed, wig, ofst, ..)
- genome : annotation files of the experiment (fasta genome, index, gtf, txdb)
- organism : name (for automatic GO, sequence analysis..)
- description : and author information (list.experiments(), show all experiments you have made with ORFik, easy to find and load them later)
- API : ORFik supports a rich API for using the experiment, like outputLibs(experiment, type = "wig") will load all libraries converted to wig format into R, loadTxdb(experiment) will load the txdb (gtf) of experiment, transcriptWindow() will automatically plot metacoverage of all libraries in the experiment, countTable(experiment) will load count tables, etc..)
- Safety : It is also a safety in that it verifies your experiments contain no duplicate, empty or non-accessible files.

Act as a way of extension of `SummarizedExperiment` by allowing more ease to find not only counts, but rather information about libraries, and annotation, so that more tasks are possible. Like coverage per position in some transcript etc.

## Constructor:
Simplest way to make is to call:
create.experiment(dir)
On some folder with NGS libraries (usually bam files) and see what you get. Some of the fields might be needed to fill in manually. Each resulting row must be unique (not including filepath, they are always unique), that means if it has replicates then that must be said explicit. And all filepaths must be unique and have files with size > 0.

Here all the columns in the experiment will be described: name (column info): examples

**libtype** library type: rna-seq, ribo-seq, CAGE etc

**stage** stage or tissue: 64cell, Shield, HEK293

**rep** replicate: 1,2,3 etc

**condition** treatment or condition: : WT (wild-type), control, target, mzdicer, starved

**fraction** fraction of total: 18, 19 (TCP / RCP fractions), or other ways to split library.

**filepath** Full filepath to file

**reverse** optional: 2nd filepath or info, only used if paired files

## Details

Special rules:
Supported:
Single/paired end bam, bed, wig, ofst + compressions of these
The reverse column of the experiments says "paired-end" if bam file. If a pair of wig files, forward and reverse strand, reverse is filepath to '-' strand wig file. Paired forward / reverse wig files, must have same name except _forward / _reverse in name
Paired end bam, when creating experiment, set pairedEndBam = c(T, T, T, F). For 3 paired end libraries, then one single end.
Naming: Will try to guess naming for tissues / stages, replicates etc. If it finds more than one hit for one file, it will not guess. Always check that it guessed correctly.

## Value

a ORFik experiment

## See Also

Other ORFik_experiment: ORFik.template.experiment(), ORFik.template.experiment.zf(), bamVarName(), create.experiment(), filepath(), libraryTypes(), organism,experiment-method, outputLibs(), read.experiment(), save.experiment(), validateExperiments()

## Examples

```
## To see an internal ORFik example
df <- ORFik.template.experiment()
## See libraries in experiment
df
## See organism of experiment
organism(df)
## See file paths in experiment
filepath(df, "default")
## Output NGS libraries in R, to .GlobalEnv
#outputLibs(df)
## Output cds of experiment annotation
#loadRegion(df, "cds")

## This is how to make it:
## Not run:
library(ORFik)

# 1. Update path to experiment data  directory (bam, bed, wig files etc)
exp_dir = "/data/processed_data/RNA-seq/Lee_zebrafish_2013/aligned/"

# 2. Set a short character name for experiment, (Lee et al 2013 -> Lee13, etc)
exper_name = "Lee13"

# 3. Create a template experiment (gtf and fasta genome)
temp <- create.experiment(exp_dir, exper_name, saveDir = NULL,
 txdb = "/data/references/Zv9_zebrafish/Danio_rerio.Zv9.79.gtf",
 fa = "/data/references/Zv9_zebrafish/Danio_rerio.Zv9.fa",
```

```
 organism = "Homo sapiens")

# 4. Make sure each row(sample) is unique and correct
# You will get a view open now, check the data.frame that it is correct:
# library type (RNA-seq, Ribo-seq), stage, rep, condition, fraction.
# Let say it did not figure out it is RNA-seq, then we do:"

temp[5:6, 1] <- "RNA" # [row 5 and 6, col 1] are library types

# You can also do this in your spread sheet program (excel, libre office)
# Now save new version, if you did not use spread sheet.
saveName <- paste0("/data/processed_data/experiment_tables_for_R/",
 exper_name,".csv")
save.experiment(temp, saveName)

# 5. Load experiment, this will validate that you actually made it correct
df <- read.experiment(saveName)

# Set experiment name not to be assigned in R variable names
df@expInVarName <- FALSE
df

## End(Not run)
```

---

`experiment.colors` *Decide color for libraries by grouping*

---

## Description

Pick the grouping wanted for colors, by default only group by libtype. Like RNA-seq(skyblue4) and Ribo-seq(orange).

## Usage

```
experiment.colors(
  df,
  color_list = "default",
  skip.libtype = FALSE,
  skip.stage = TRUE,
  skip.replicate = TRUE,
  skip.fraction = TRUE,
  skip.condition = TRUE
)
```

## Arguments

| | |
|---|---|
| `df` | an ORFik `experiment` |

| | |
|---|---|
| color_list | a character vector of colors, default "default". That is the vector c("skyblue4", 'orange', "green", "red", "gray", "yellow", "blue", "red2", "orange3"). Picks number of colors needed to make groupings have unique color |
| skip.libtype | a logical (FALSE), don't include libtype |
| skip.stage | a logical (FALSE), don't include stage in variable name. |
| skip.replicate | a logical (FALSE), don't include replicate in variable name. |
| skip.fraction | a logical (FALSE), don't include fraction |
| skip.condition | a logical (FALSE), don't include condition in variable name. |

## Value

a character vector of colors

---

# export.bed12 *Export as bed12 format*

---

## Description

bed format for multiple exons per group, as transcripts. Can be use as alternative as a sparse .gff format for ORFs. Can be direct input for ucsc browser or IGV

## Usage

```
export.bed12(grl, file, rgb = 0)
```

## Arguments

| | |
|---|---|
| grl | A GRangesList |
| file | a character path to valid output file name |
| rgb | integer vector, default (0), either single integer or vector of same size as grl to specify groups. It is adviced to not use more than 8 different groups |

## Details

If grl has no names, groups will be named 1,2,3,4..

## Value

NULL (File is saved as .bed)

## See Also

Other utils: bedToGR(), convertToOneBasedRanges(), export.bigWig(), export.fstwig(), export.wiggle(), fimport(), findFa(), fread.bed(), optimizeReads(), readBam(), readBigWig(), readWig()

## Examples

```
grl <- GRangesList(GRanges("1", c(1,3,5), "+"))
# export.bed12(grl, "output/path/orfs.bed")
```

---

`export.bedo` *Store GRanges object as .bedo*

---

## Description

.bedo is .bed ORFik, an optimized bed format for coverage reads with read lengths .bedo is a text based format with columns (6 maximum):
1. chromosome
2. start
3. end
4. strand
5. ref width (cigar # M's, match/mismatch total)
6. duplicates of that read

## Usage

```
export.bedo(object, out)
```

## Arguments

| | |
|---|---|
| `object` | a GRanges object |
| `out` | a character, location on disc (full path) |

## Details

Positions are 1-based, not 0-based as .bed. End will be removed if all ends equals all starts. Import with import.bedo

## Value

NULL, object saved to disc

`export.bedoc` *Store GAlignments object as .bedoc*

## Description

A fast way to store, load and use bam files. (we now recommend using `link{export.ofst}` instead!)
.bedoc is .bed ORFik, an optimized bed format for coverage reads with cigar and replicate number.
.bedoc is a text based format with columns (5 maximum):
1. chromosome
2. cigar: (cigar # M's, match/mismatch total)
3. start (left most position)
4. strand (+, -, *)
5. score: duplicates of that read

## Usage

```
export.bedoc(object, out)
```

## Arguments

| | |
|---|---|
| `object` | a GAlignments object |
| `out` | a character, location on disc (full path) |

## Details

Positions are 1-based, not 0-based as .bed. Import with import.bedoc

## Value

NULL, object saved to disc

`export.bigWig` *Export as bigWig format*

## Description

Will create 2 files, 1 for + strand (*_forward.bigWig) and 1 for - strand (*_reverse.bigWig). If all ranges are * stranded, will output 1 file. Can be direct input for ucsc browser or IGV

## Usage

```
export.bigWig(
  x,
  file,
  split.by.strand = TRUE,
  is_pre_collapsed = FALSE,
  seq_info = seqinfo(x)
)
```

## Arguments

| | |
|---|---|
| x | A GRangesList, GAlignment GAlignmentPairs with score column. Will be converted to 5' end position of original range. If score column does not exist, will group ranges and give replicates as score column. Since bigWig needs a score column to represent counts! |
| file | a character path to valid output file name |
| split.by.strand | logical, default TRUE. Split bigWig into 2 files, one for each strand. |
| is_pre_collapsed | logical, default FALSE. Have you already collapsed reads with collapse.by.scores, so each positions is only in 1 GRanges object with a score column per readlength? Set to TRUE, only if you are sure, will give a speedup. |
| seq_info | a Seqinfo object, default seqinfo(x). Must have non NA seqlengths defined! |

## Value

invisible(NULL) (File is saved as 2 .bigWig files)

## References

https://genome.ucsc.edu/goldenPath/help/bigWig.html

## See Also

Other utils: bedToGR(), convertToOneBasedRanges(), export.bed12(), export.fstwig(), export.wiggle(), fimport(), findFa(), fread.bed(), optimizeReads(), readBam(), readBigWig(), readWig()

## Examples

```
x <- c(GRanges("1", c(1,3,5), "-"), GRanges("1", c(1,3,5), "+"))
seqlengths(x) <- 10
file <- file.path(tempdir(), "rna.bigWig")
# export.bigWig(x, file)
# export.bigWig(covRleFromGR(x), file)
```

`export.fstwig` *Export as fstwig (fastwig) format*

## Description

Will create 2 files, 1 for + strand (*_forward.fstwig) and 1 for - strand (*_reverse.fstwig). If all ranges are * stranded, will output 1 file.

## Usage

```
export.fstwig(
  x,
  file,
  by.readlength = TRUE,
  by.chromosome = TRUE,
  compress = 50
)
```

## Arguments

| | |
|---|---|
| `x` | A GRangesList, GAlignment GAlignmentPairs with score column or coverage RLElist Will be converted to 5' end position of original range. If score column does not exist, will group ranges and give replicates as score column. |
| `file` | a character path to valid output file name |
| `by.readlength` | logical, default TRUE |
| `by.chromosome` | logical, default TRUE |
| `compress` | value in the range 0 to 100, indicating the amount of compression to use. Lower values mean larger file sizes. The default compression is set to 50. |

## Value

invisible(NULL) (File is saved as 2 .fstwig files)

## References

"TODO"

## See Also

Other utils: `bedToGR()`, `convertToOneBasedRanges()`, `export.bed12()`, `export.bigWig()`, `export.wiggle()`, `fimport()`, `findFa()`, `fread.bed()`, `optimizeReads()`, `readBam()`, `readBigWig()`, `readWig()`

## Examples

```
x <- c(GRanges("1", c(1,3,5), "-"), GRanges("1", c(1,3,5), "+"))
x$size <- rep(c(28, 29), length.out = length(x))
x$score <- c(5,1,2,5,1,6)
seqlengths(x) <- 5
# export.fstwig(x, "~/Desktop/ribo")
```

---

`export.ofst` *Store GRanges / GAlignments object as .ofst*

---

## Description

A much faster way to store, load and use bam files.
.ofst is ORFik fast serialized object, an optimized format for coverage reads with cigar and replicate number. It uses the fst format as back-end: `fst-package`.
A .ofst ribo seq file can compress the information in a bam file from 5GB down to a few MB. This new files has super fast reading time, only a few seconds, instead of minutes. It also has random index access possibility of the file.
.ofst is represented as a data.frane format with minimum 4 columns:
1. chromosome
2. start (left most position)
3. strand (+, -, *)
4. width (not added if cigar exists)
5. cigar (not needed if width exists): (cigar # M's, match/mismatch total)
5. score: duplicates of that read
6. size: qwidth according to reference of read

If file is from GAlignmentPairs, it will contain a cigar1, cigar2 instead of cigar and start1 and start2 instead of start

## Usage

```
export.ofst(x, file, ...)
```

## Arguments

| | |
|---|---|
| `x` | a GRanges, GAlignments or GAlignmentPairs object |
| `file` | a character, location on disc (full path) |
| `...` | additional arguments for write_fst |

## Details

Other columns can be named whatever you want and added to meta columns. Positions are 1-based, not 0-based as .bed. Import with import.ofst

## Value

NULL, object saved to disc

## Examples

```
## GRanges
gr <- GRanges("1:1-3:-")
# export.ofst(gr, file = "path.ofst")
## GAlignment
# Make input data.frame
df <- data.frame(seqnames = "1", cigar = "3M", start = 1L, strand = "+")
ga <- ORFik:::getGAlignments(df)
# export.ofst(ga, file = "path.ofst")
```

---

`export.ofst,GAlignmentPairs-method`

*Store GRanges / GAlignments object as .ofst*

---

## Description

A much faster way to store, load and use bam files.
.ofst is ORFik fast serialized object, an optimized format for coverage reads with cigar and replicate number. It uses the fst format as back-end: `fst-package`.
A .ofst ribo seq file can compress the information in a bam file from 5GB down to a few MB. This new files has super fast reading time, only a few seconds, instead of minutes. It also has random index access possibility of the file.
.ofst is represented as a data.frane format with minimum 4 columns:
1. chromosome
2. start (left most position)
3. strand (+, -, *)
4. width (not added if cigar exists)
5. cigar (not needed if width exists): (cigar # M's, match/mismatch total)
5. score: duplicates of that read
6. size: qwidth according to reference of read

If file is from GAlignmentPairs, it will contain a cigar1, cigar2 instead of cigar and start1 and start2 instead of start

## Usage

```
## S4 method for signature 'GAlignmentPairs'
export.ofst(x, file, ...)
```

## Arguments

| | |
|---|---|
| `x` | a GRanges, GAlignments or GAlignmentPairs object |
| `file` | a character, location on disc (full path) |
| `...` | additional arguments for write_fst |

### Details

Other columns can be named whatever you want and added to meta columns. Positions are 1-based, not 0-based as .bed. Import with import.ofst

### Value

NULL, object saved to disc

### Examples

```
## GRanges
gr <- GRanges("1:1-3:-")
# export.ofst(gr, file = "path.ofst")
## GAlignment
# Make input data.frame
df <- data.frame(seqnames = "1", cigar = "3M", start = 1L, strand = "+")
ga <- ORFik:::getGAlignments(df)
# export.ofst(ga, file = "path.ofst")
```

---

## export.ofst,GAlignments-method

*Store GRanges / GAlignments object as .ofst*

---

### Description

A much faster way to store, load and use bam files.
.ofst is ORFik fast serialized object, an optimized format for coverage reads with cigar and replicate number. It uses the fst format as back-end: fst-package.
A .ofst ribo seq file can compress the information in a bam file from 5GB down to a few MB. This new files has super fast reading time, only a few seconds, instead of minutes. It also has random index access possibility of the file.
.ofst is represented as a data.frane format with minimum 4 columns:
1. chromosome
2. start (left most position)
3. strand (+, -, *)
4. width (not added if cigar exists)
5. cigar (not needed if width exists): (cigar # M's, match/mismatch total)
5. score: duplicates of that read
6. size: qwidth according to reference of read

If file is from GAlignmentPairs, it will contain a cigar1, cigar2 instead of cigar and start1 and start2 instead of start

### Usage

```
## S4 method for signature 'GAlignments'
export.ofst(x, file, ...)
```

### Arguments

| | |
|---|---|
| x | a GRanges, GAlignments or GAlignmentPairs object |
| file | a character, location on disc (full path) |
| ... | additional arguments for write_fst |

### Details

Other columns can be named whatever you want and added to meta columns. Positions are 1-based, not 0-based as .bed. Import with import.ofst

### Value

NULL, object saved to disc

### Examples

```
## GRanges
gr <- GRanges("1:1-3:-")
# export.ofst(gr, file = "path.ofst")
## GAlignment
# Make input data.frame
df <- data.frame(seqnames = "1", cigar = "3M", start = 1L, strand = "+")
ga <- ORFik:::getGAlignments(df)
# export.ofst(ga, file = "path.ofst")
```

---

## export.ofst,GRanges-method

*Store GRanges / GAlignments object as .ofst*

---

### Description

A much faster way to store, load and use bam files.
.ofst is ORFik fast serialized object, an optimized format for coverage reads with cigar and replicate number. It uses the fst format as back-end: fst-package.
A .ofst ribo seq file can compress the information in a bam file from 5GB down to a few MB. This new files has super fast reading time, only a few seconds, instead of minutes. It also has random index access possibility of the file.
.ofst is represented as a data.frane format with minimum 4 columns:
1. chromosome
2. start (left most position)
3. strand (+, -, *)
4. width (not added if cigar exists)
5. cigar (not needed if width exists): (cigar # M's, match/mismatch total)
5. score: duplicates of that read
6. size: qwidth according to reference of read

If file is from GAlignmentPairs, it will contain a cigar1, cigar2 instead of cigar and start1 and start2 instead of start

### Usage

```
## S4 method for signature 'GRanges'
export.ofst(x, file, ...)
```

### Arguments

| | |
|---|---|
| x | a GRanges, GAlignments or GAlignmentPairs object |
| file | a character, location on disc (full path) |
| ... | additional arguments for write_fst |

### Details

Other columns can be named whatever you want and added to meta columns. Positions are 1-based, not 0-based as .bed. Import with import.ofst

### Value

NULL, object saved to disc

### Examples

```
## GRanges
gr <- GRanges("1:1-3:-")
# export.ofst(gr, file = "path.ofst")
## GAlignment
# Make input data.frame
df <- data.frame(seqnames = "1", cigar = "3M", start = 1L, strand = "+")
ga <- ORFik:::getGAlignments(df)
# export.ofst(ga, file = "path.ofst")
```

---

## export.wiggle *Export as wiggle format*

---

### Description

Will create 2 files, 1 for + strand (*_forward.wig) and 1 for - strand (*_reverse.wig). If all ranges are * stranded, will output 1 file. Can be direct input for ucsc browser or IGV

### Usage

```
export.wiggle(x, file)
```

### Arguments

| | |
|---|---|
| x | A GRangesList, GAlignment GAlignmentPairs with score column. Will be converted to 5' end position of original range. If score column does not exist, will group ranges and give replicates as score column. |
| file | a character path to valid output file name |

## Value

invisible(NULL) (File is saved as 2 .wig files)

## References

https://genome.ucsc.edu/goldenPath/help/wiggle.html

## See Also

Other utils: bedToGR(), convertToOneBasedRanges(), export.bed12(), export.bigWig(), export.fstwig(), fimport(), findFa(), fread.bed(), optimizeReads(), readBam(), readBigWig(), readWig()

## Examples

```
x <- c(GRanges("1", c(1,3,5), "-"), GRanges("1", c(1,3,5), "+"))
# export.wiggle(x, "output/path/rna.wig")
```

---

# extendLeaders *Extend the leaders transcription start sites.*

---

## Description

Will extend the leaders or transcripts upstream (5' end) by extension. The extension is general not relative, that means splicing will not be taken into account. Requires the grl to be sorted beforehand, use sortPerGroup to get sorted grl.

## Usage

```
extendLeaders(
  grl,
  extension = 1000L,
  cds = NULL,
  is.circular = all(isCircular(grl) %in% TRUE)
)
```

## Arguments

| | |
|---|---|
| grl | usually a GRangesList of 5' utrs or transcripts. Can be used for any extension of groups. |
| extension | an integer, how much to extend upstream (5' end). Eiter single value that will apply for all, or same as length of grl which will give 1 update value per grl object. Or a GRangesList where start / stops by strand are the positions to use as new starts. |
| cds | a GRangesList of coding sequences, 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. |

is.circular logical, default FALSE if not any is: all(isCircular(grl) Where grl is the ranges checked. If TRUE, allow ranges to extend below position 1 on chromosome. Since circular genomes can have negative coordinates.

## Value

an extended GRangeslist

## See Also

Other ExtendGenomicRanges: asTX(), coveragePerTiling(), extendTrailers(), reduceKeepAttr(), tile1(), txSeqsFromFa(), windowPerGroup()

## Examples

```
library(GenomicFeatures)
samplefile <- system.file("extdata", "hg19_knownGene_sample.sqlite",
                          package = "GenomicFeatures")
txdb <- loadDb(samplefile)
fiveUTRs <- fiveUTRsByTranscript(txdb, use.names = TRUE) # <- extract only 5' leaders
tx <- exonsBy(txdb, by = "tx", use.names = TRUE)
cds <- cdsBy(txdb,"tx",use.names = TRUE)
## extend leaders upstream 1000
extendLeaders(fiveUTRs, extension = 1000)
## now try(extend upstream 1000, add all 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)
## Circular genome (allow negative coordinates)
circular_fives <- fiveUTRs
isCircular(circular_fives) <- rep(TRUE, length(isCircular(circular_fives)))
extendLeaders(circular_fives, extension = 32672841L)
```

---

# extendTrailers *Extend the Trailers transcription stop sites*

---

## Description

Will extend the trailers or transcripts downstream (3' end) by extension. The extension is general not relative, that means splicing will not be taken into account. Requires the grl to be sorted beforehand, use sortPerGroup to get sorted grl.

## Usage

```
extendTrailers(
  grl,
  extension = 1000L,
  is.circular = all(isCircular(grl) %in% TRUE)
)
```

## Arguments

| | |
|---|---|
| grl | usually a GRangesList of 3' utrs or transcripts. Can be used for any extension of groups. |
| extension | an integer, how much to extend downstream (3' end). Eiter single value that will apply for all, or same as length of grl which will give 1 update value per grl object. Or a GRangesList where start / stops sites by strand are the positions to use as new starts. |
| is.circular | logical, default FALSE if not any is: all(isCircular(grl) Where grl is the ranges checked. If TRUE, allow ranges to extend below position 1 on chromosome. Since circular genomes can have negative coordinates. |

## Value

an extended GRangeslist

## See Also

Other ExtendGenomicRanges: asTX(), coveragePerTiling(), extendLeaders(), reduceKeepAttr(), tile1(), txSeqsFromFa(), windowPerGroup()

## Examples

```
library(GenomicFeatures)
samplefile <- system.file("extdata", "hg19_knownGene_sample.sqlite",
                          package = "GenomicFeatures")
txdb <- loadDb(samplefile)
threeUTRs <- threeUTRsByTranscript(txdb) # <- extract only 5' leaders
tx <- exonsBy(txdb, by = "tx", use.names = TRUE)
## now try(extend downstream 1000):
extendTrailers(threeUTRs, extension = 1000)
## Or on transcripts
extendTrailers(tx, extension = 1000)
## Circular genome (allow negative coordinates)
circular_three <- threeUTRs
isCircular(circular_three) <- rep(TRUE, length(isCircular(circular_three)))
extendTrailers(circular_three, extension = 126200008L)[41] # <- negative stop coordinate
```

## extract_run_id *Extract SRR/ERR/DRR run IDs from string*

### Description

Extract SRR/ERR/DRR run IDs from string

### Usage

```
extract_run_id(
  x,
  search = "(SRR[0-9]+|DRR[0-9]+|ERR[0-9]+)",
  only_valid = FALSE
)
```

### Arguments

| | |
|---|---|
| x | character vector to search through. |
| search | the regex search, default: "(SRR[0-9]+\|DRR[0-9]+\|ERR[0-9]+)" |
| only_valid | logical, default FALSE. If TRUE, return only the hits. |

### Value

a character vector of run accepted run ids according to search, if only_valid named character vector for which indices are returned

### Examples

```
search <- c("SRR1230123_absdb", "SRR1241204124_asdasd", "asd_ERR1231230213",
 "DRR12412412_asdqwe", "ASDASD_ASDASD", "SRRASDASD")
ORFik:::extract_run_id(search)
ORFik:::extract_run_id(search, only_valid = TRUE)
```

## f *strandMode covRle*

### Description

strandMode covRle

### Usage

```
f(x)
```

### Arguments

| | |
|---|---|
| x | a covRle object |

### Value

the forward RleList

---

## f,covRle-method *strandMode covRle*

---

### Description

strandMode covRle

### Usage

```
## S4 method for signature 'covRle'
f(x)
```

### Arguments

| | |
|---|---|
| x | a covRle object |

### Value

the forward RleList

---

## filepath *Get filepaths to ORFik experiment*

---

### Description

If other type than "default" is given and that type is not found (and 'fallback' is TRUE), it will return you ofst files, if they do not exist, then default filepaths without warning.

### Usage

```
filepath(
  df,
  type,
  basename = FALSE,
  fallback = type %in% c("pshifted", "bed", "ofst", "bedoc", "bedo"),
  suffix_stem = "AUTO",
  base_folders = libFolder(df)
)
```

## Arguments

| | |
|---|---|
| df | an ORFik experiment |
| type | a character(default: "default"), load files in experiment or some precomputed variant, like "ofst" or "pshifted". These are made with ORFik:::convertLibs(), shiftFootprintsByExperiment(), etc. Can also be custom user made folders inside the experiments bam folder. It acts in a recursive manner with priority: If you state "pshifted", but it does not exist, it checks "ofst". If no .ofst files, it uses "default", which always must exists.<br>Presets are (folder is relative to default lib folder, some types fall back to other formats if folder does not exist):<br>- "default": load the original files for experiment, usually bam.<br>- "ofst": loads ofst files from the ofst folder, relative to lib folder (falls back to default)<br>- "pshifted": loads ofst, wig or bigwig from pshifted folder (falls back to ofst, then default)<br>- "cov": Load covRle objects from cov_RLE folder (fail if not found)<br>- "covl": Load covRleList objects, from cov_RLE_List folder (fail if not found)<br>- "bed": Load bed files, from bed folder (falls back to default)<br>- Other formats must be loaded directly with fimport |
| basename | logical, default (FALSE). Get relative paths instead of full. Only use for inspection! |
| fallback | logical, default: type If TRUE, will use type fallback, see above for info. |
| suffix_stem | character, default "AUTO". Which is "" for all except type = "pshifted". Then it is "_pshifted" appended to end of names before format. Can be vector, then it searches suffixes in priority: so if you insert c("_pshifted", ""), it will look for suffix _pshifted, then the empty suffix. |
| base_folders | character vector, default libFolder(df), path to base folder to search for library variant directories. If single path (length == 1), it will apply to all libraries in df. If df is a collection, an experiment where libraries are put in different folders and library variants like pshifted are put inside those respective folders, set base_folders = libFolder(df, mode = "all") |

## Details

For pshifted libraries, if "pshifted" is specified as type: if if multiple formats exist it will use a priority: ofst -> bigwig -> wig -> bed. For formats outside default, all files must be stored in the directory of the first file: `base_folder <- libFolder(df)`

## Value

a character vector of paths, or a list of character with 2 paths per, if paired libraries exists

## See Also

Other ORFik_experiment: `ORFik.template.experiment()`, `ORFik.template.experiment.zf()`, `bamVarName()`, `create.experiment()`, `experiment-class`, `libraryTypes()`, `organism,experiment-method`, `outputLibs()`, `read.experiment()`, `save.experiment()`, `validateExperiments()`

## Examples

```
df <- ORFik.template.experiment()
filepath(df, "default")
# Subset
filepath(df[9,], "default")
# Other format path
filepath(df[9,], "ofst")
## If you have pshifted files, see shiftFootprintsByExperiment()
filepath(df[9,], "pshifted") # <- falls back to ofst
```

---

`filterExtremePeakGenes` *Filter out transcript by a median filter*

---

## Description

For removing very extreme peaks in coverage plots, use high quantiles, like 99. Used to make your plots look better, by removing extreme peaks.

## Usage

```
filterExtremePeakGenes(
  tx,
  reads,
  upstream = NULL,
  downstream = NULL,
  multiplier = "0.99",
  min_cutoff = "0.999",
  pre_filter_minimum = 0,
  average = "median"
)
```

## Arguments

| | |
|---|---|
| `tx` | a GRangesList |
| `reads` | a GAlignments or GRanges |
| `upstream` | numeric or NULL, default NULL. if you want window of tx, instead of whole, specify how much upstream from start of tx, 10 is include 10 bases before start |
| `downstream` | numeric or NULL, default NULL. if you want window of tx, instead of whole, specify how much downstream from start of tx, 10 is go 10 bases into tx from start. |
| `multiplier` | a character or numeric, default "0.99", either a quantile if input is string[0-1], like "0.99", or numeric value if input is numeric. How much bigger than median / mean counts per gene, must a value be to be defined as extreme ? |
| `min_cutoff` | a character or numeric, default "0.999", either a quantile if input is string[0-1], like "0.999", or numeric value if input is numeric. Lowest allowed value |

pre_filter_minimum
: numeric, default 0. If value is x, will remove all positions in all genes with coverage < x, before median filter is applied. Set to 1 to remove all 0 positions.

average
: character, default "median". Alternative: "mean". How to scale the multiplier argument, from median or mean of gene coverage.

## Value

GRangesList (filtered)

---

`filterTranscripts` *Filter transcripts by lengths*

---

## Description

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

## Usage

```
filterTranscripts(
  txdb,
  minFiveUTR = 30L,
  minCDS = 150L,
  minThreeUTR = 30L,
  longestPerGene = TRUE,
  stopOnEmpty = TRUE,
  by = "tx",
  create.fst.version = FALSE
)
```

## Arguments

txdb
: a TxDb file or a path to one of: (.gtf ,.gff, .gff2, .gff2, .db or .sqlite), if it is a GRangesList, it will return it self.

minFiveUTR
: (integer) minimum bp for 5' UTR during filtering for the transcripts. Set to NULL if no 5' UTRs exists for annotation.

minCDS
: (integer) minimum bp for CDS during filtering for the transcripts

minThreeUTR
: (integer) minimum bp for 3' UTR during filtering for the transcripts. Set to NULL if no 3' UTRs exists for annotation.

longestPerGene
: logical (TRUE), return only longest valid transcript per gene. NOTE: This is by priority longest cds isoform, if equal then pick longest total transcript. So if transcript is shorter but cds is longer, it will still be the one returned.

stopOnEmpty
: logical TRUE, stop if no valid transcripts are found ?

by a character, default "tx" Either "tx" or "gene". What names to output region by, the transcript name "tx" or gene names "gene". NOTE: this is not the same as cdsBy(txdb, by = "gene"), cdsBy would then only give 1 cds per Gene, loadRegion gives all isoforms, but with gene names.

create.fst.version logical, FALSE. If TRUE, creates a .fst version of the transcript length table (if it not already exists), reducing load time from ~ 15 seconds to ~ 0.01 second next time you run filterTranscripts with this txdb object. The file is stored in the same folder as the genome this txdb is created from, with the name: `paste0(ORFik:::remove.file_ext(metadata(txdb)[3,2]), "_", gsub(" \(.*|
|:", "", metadata(txdb)[metadata(txdb)[,1] == "Creation time",2]), "_txLengths.fst")` Some error checks are done to see this is a valid location, if the txdb data source is a repository like UCSC and not a local folder, it will not be made.

## Details

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

If your annotation does not have leaders or trailers, set them to NULL, since 0 means there must exist a column called utr3_len etc. Genes with gene_id = NA will be be removed.

## Value

a character vector of valid transcript names

## Examples

```
df <- ORFik.template.experiment.zf()
txdb <- loadTxdb(df)
txNames <- filterTranscripts(txdb, minFiveUTR = 1, minCDS = 30,
                             minThreeUTR = 1)
loadRegion(txdb, "mrna")[txNames]
loadRegion(txdb, "5utr")[txNames]
```

---

# fimport *Load any type of sequencing reads*

---

## Description

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

## Usage

```
fimport(path, chrStyle = NULL, param = NULL, strandMode = 0)
```

## Arguments

| | |
|---|---|
| path | a character path to file (1 or 2 files), or data.table with 2 colums(forward&reverse) or a GRanges/Galignment/GAlignmentPairs object etc. If it is ranged object it will presume to be already loaded, so will return the object as it is, updating the seqlevelsStyle if given. |
| chrStyle | a GRanges object, TxDb, FaFile, , a seqlevelsStyle or Seqinfo. (Default: NULL) to get seqlevelsStyle from. In addition if it is a Seqinfo object, seqinfo will be updated. Example of seqlevelsStyle update: Is chromosome 1 called chr1 or 1, is mitocondrial chromosome called MT or chrM etc. Will use 1st seqlevel-style if more are present. Like: c("NCBI", "UCSC") -> pick "NCBI" |
| param | NULL or a ScanBamParam object. Like for scanBam, this influences what fields and which records are imported. However, note that the fields specified thru this ScanBamParam object will be loaded *in addition* to any field required for generating the returned object (GAlignments, GAlignmentPairs, or GappedReads object), but only the fields requested by the user will actually be kept as metadata columns of the object.<br>By default (i.e. param=NULL or param=ScanBamParam()), no additional field is loaded. The flag used is scanBamFlag(isUnmappedQuery=FALSE) for readGAlignments, readGAlignmentsList, and readGappedReads. (i.e. only records corresponding to mapped reads are loaded), and scanBamFlag(isUnmappedQuery=FALSE, isPaired=TRUE, hasUnmappedMate=FALSE) for readGAlignmentPairs (i.e. only records corresponding to paired-end reads with both ends mapped are loaded). |
| strandMode | numeric, default 0. Only used for paired end bam files. One of (0: strand = *, 1: first read of pair is +, 2: first read of pair is -). See ?strandMode. Note: Sets default to 0 instead of 1, as readGAlignmentPairs uses 1. This is to guarantee hits, but will also make mismatches of overlapping transcripts in opposite directions. |

## Details

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

## Value

a GAlignments/GRanges object, depending on input.

## See Also

Other utils: bedToGR(), convertToOneBasedRanges(), export.bed12(), export.bigWig(), export.fstwig(), export.wiggle(), findFa(), fread.bed(), optimizeReads(), readBam(), readBigWig(), readWig()

## Examples

```
bam_file <- system.file("extdata/Danio_rerio_sample", "ribo-seq.bam", package = "ORFik")
fimport(bam_file)
# Certain chromosome naming
fimport(bam_file, "NCBI")
```

```
# Paired end bam strandMode 1:
fimport(bam_file, strandMode = 1)
# (will have no effect in this case, since it is not paired end)
```

---

findFa *Convenience wrapper for Rsamtools FaFile*

---

## Description

Get fasta file object, to find sequences in file.
Will load and import file if necessarry.

## Usage

```
findFa(faFile)
```

## Arguments

| | |
|---|---|
| faFile | FaFile, BSgenome, fasta/index file path or an ORFik experiment. This file is usually used to find the transcript sequences from some GRangesList. |

## Value

a FaFile or BSgenome

## See Also

Other utils: bedToGR(), convertToOneBasedRanges(), export.bed12(), export.bigWig(), export.fstwig(), export.wiggle(), fimport(), fread.bed(), optimizeReads(), readBam(), readBigWig(), readWig()

## Examples

```
# Some fasta genome with existing fasta index in same folder
path <- system.file("extdata/references/danio_rerio", "genome_dummy.fasta", package = "ORFik")
findFa(path)
```

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

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| `grl` | A `GRangesList` of the original sequences that gave the orfs in Genomic coordinates. If grl_is_sorted = TRUE (default), negative exon ranges per grl object must be sorted in descending orders. |
| `seqs` | (DNAStringSet or character vector) - DNA/RNA sequences to search for Open Reading Frames. Can be both uppercase or lowercase. Easiest call to get seqs if you want only regions from a fasta/fasta index pair is: seqs = ORFik:::txSeqsFromFa(grl, faFile), where grl is a GRanges/List of search regions and faFile is a `FaFile`. Note: Remember that if you extracted through a GRanges object, that must have been sorted with negative strand exons descending. |
| `startCodon` | (character vector) Possible START codons to search for. Check `startDefinition` for helper function. Note that it is case sensitive, so "atg" would give 0 hits for a sequence with only capital "ATG" ORFs. |
| `stopCodon` | (character vector) Possible STOP codons to search for. Check `stopDefinition` for helper function. Note that it is case sensitive, so "tga" would give 0 hits for a sequence with only capital "TGA" ORFs. |
| `longestORF` | (logical) Default TRUE. Keep only the longest ORF per unique stopcodon: (seqname, strand, stopcodon) combination, Note: Not longest per transcript! You can also use function `longestORFs` after creation of ORFs for same result. |
| `minimumLength` | (integer) Default is 0. Which is START + STOP = 6 bp. Minimum length of ORF, without counting 3bps 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 (default: FALSE), should output GRangesList be grouped by exons per ORF (TRUE) or by orfs per transcript (FALSE)? |
| grl_is_sorted | logical, default FALSE If FALSE will sort negative transcript in descending order for you. If you loaded ranges with default methods this is already the case, so you can set to TRUE to save some time. |

## Details

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

See vignette for real life example.

## Value

A GRangesList of ORFs.

## See Also

Other findORFs: `findORFs()`, `findORFsFasta()`, `findUORFs()`, `startDefinition()`, `stopDefinition()`

## Examples

```
# First show simple example using findORFs
# This sequence has ORFs at 1-9 and 4-9
seqs <- DNAStringSet("ATGATGTAA") # the dna transcript sequence
findORFs(seqs)
# lets assume that this sequence comes from two exons as follows
# Then we need to use findMapORFs instead of findORFs,
#  for splicing information
gr <- GRanges(seqnames = "1", # chromosome 1
              ranges = IRanges(start = c(21, 10), end = c(23, 15)),
              strand = "-", #
              names = "tx1") #From transcript 1 on chr 1
grl <- GRangesList(tx1 = gr) # 1 transcript with 2 exons
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))
# More advanced example and how to save sequences found in vignette
```

findORFs *Find Open Reading Frames.*

## Description

Find all Open Reading Frames (ORFs) on the simple input sequences in ONLY 5'- 3' direction (+), but within all three possible reading frames. Do not use findORFs for mapping to full chromosomes, then use findMapORFs! For each sequence of the input vector IRanges with START and STOP positions (inclusive) will be returned as IRangesList. Returned coordinates are relative to the input sequences.

## Usage

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

## Arguments

| | |
|---|---|
| seqs | (DNAStringSet or character vector) - DNA/RNA sequences to search for Open Reading Frames. Can be both uppercase or lowercase. Easiest call to get seqs if you want only regions from a fasta/fasta index pair is: seqs = ORFik:::txSeqsFromFa(grl, faFile), where grl is a GRanges/List of search regions and faFile is a FaFile. Note: Remember that if you extracted through a GRanges object, that must have been sorted with negative strand exons descending. |
| startCodon | (character vector) Possible START codons to search for. Check startDefinition for helper function. Note that it is case sensitive, so "atg" would give 0 hits for a sequence with only capital "ATG" ORFs. |
| stopCodon | (character vector) Possible STOP codons to search for. Check stopDefinition for helper function. Note that it is case sensitive, so "tga" would give 0 hits for a sequence with only capital "TGA" ORFs. |
| longestORF | (logical) Default TRUE. Keep only the longest ORF per unique stopcodon: (seqname, strand, stopcodon) combination, Note: Not longest per transcript! You can also use function longestORFs after creation of ORFs for same result. |
| minimumLength | (integer) Default is 0. Which is START + STOP = 6 bp. Minimum length of ORF, without counting 3bps for START and STOP codons. For example minimumLength = 8 will result in size of ORFs to be at least START + 8*3 (bp) + STOP = 30 bases. Use this param to restrict search. |

## Details

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

## Value

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

## See Also

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

## Examples

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

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

findORFs(c("ATGTAA", "ATGATGTAA")) # 1 ORF per transcript

## Get DNA sequences from ORFs
seq <- DNAStringSet(c("ATGTAA", "AAA", "ATGATGTAA"))
names(seq) <- c("tx1", "tx2", "tx3")
orfs <- findORFs(seq, longestORF = FALSE)

#  you can get sequences like this:
gr <- unlist(orfs, use.names = TRUE)
gr <- GRanges(seqnames = names(seq)[as.integer(names(gr))],
 ranges = gr, strand = "+")
# Give them some proper names:
names(gr) <- paste0("ORF_", seq.int(length(gr)), "_", seqnames(gr))
orf_seqs <- getSeq(seq, gr)
orf_seqs
# Save as .fasta (orf_seqs must be of type DNAStringSet)
# writeXStringSet(orf_seqs, "orfs.fasta")
## Reading from file and find ORFs
#findORFs(readDNAStringSet("path/to/transcripts.fasta"))
```

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

## Description

Should be used for procaryote genomes or transcript sequences as fasta. Makes no sence for eukaryote whole genomes, since those contains splicing (use findMapORFs for spliced ranges). 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 circular 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. Or a already loaded R object of either types: "BSgenome" or "DNAStringSet" with named sequences |
| `startCodon` | (character vector) Possible START codons to search for. Check `startDefinition` for helper function. Note that it is case sensitive, so "atg" would give 0 hits for a sequence with only capital "ATG" ORFs. |
| `stopCodon` | (character vector) Possible STOP codons to search for. Check `stopDefinition` for helper function. Note that it is case sensitive, so "tga" would give 0 hits for a sequence with only capital "TGA" ORFs. |
| `longestORF` | (logical) Default TRUE. Keep only the longest ORF per unique stopcodon: (seqname, strand, stopcodon) combination, Note: Not longest per transcript! You can also use function `longestORFs` after creation of ORFs for same result. |
| `minimumLength` | (integer) Default is 0. Which is START + STOP = 6 bp. Minimum length of ORF, without counting 3bps 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. Also make sure your fasta file is valid (no hidden spaces etc), as this might break the coordinate system!

## Value

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

## See Also

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

## Examples

```
# location of the example fasta file
example_genome <- system.file("extdata/references/danio_rerio", "genome_dummy.fasta",
 package = "ORFik")
orfs <- findORFsFasta(example_genome)
# To store ORF sequences (you need indexed genome .fai file):
fa <- FaFile(example_genome)
names(orfs) <- paste0("ORF_", seq.int(length(orfs)), "_", seqnames(orfs))
orf_seqs <- getSeq(fa, orfs)
# You sequences (fa), needs to have isCircular(fa) == TRUE for it to work
# on circular wrapping ranges!

# writeXStringSet(DNAStringSet(orf_seqs), "orfs.fasta")
```

---

# findPeaksPerGene *Find peaks per gene*

---

## Description

For finding the peaks (stall sites) per gene, with some default filters. A peak is basically a position of very high coverage compared to its surrounding area, as measured using zscore.

## Usage

```
findPeaksPerGene(
  tx,
  reads,
  top_tx = 0.5,
  min_reads_per_tx = 20,
  min_reads_per_peak = 10,
  type = "max"
)
```

## Arguments

| | |
|---|---|
| tx | a GRangesList |
| reads | a GAlignments or GRanges, must be 1 width reads like p-shifts, or other reads that is single positioned. It will work with non 1 width bases, but you then get larger areas for peaks. |
| top_tx | numeric, default 0.50 (only use 50% top transcripts by read counts). |
| min_reads_per_tx | numeric, default 20. Gene must have at least 20 reads, applied before type filter. |
| min_reads_per_peak | numeric, default 10. Peak must have at least 10 reads. |
| type | character, default "max". Get only max peak per gene. Alternatives: "all", all peaks passing the input filter will be returned. "median", only peaks that is higher than the median of all peaks. "maxmedian": get first "max", then median of those. |

## Details

For more details see reference, which uses a slightly different method by zscore of a sliding window instead of over the whole tx.

## Value

a data.table of gene_id, position, counts of the peak, zscore and standard deviation of the peak compared to rest of gene area.

## References

doi: 10.1261/rna.065235.117

## Examples

```
df <- ORFik.template.experiment()
cds <- loadRegion(df, "cds")
# Load ribo seq from ORFik
rfp <- fimport(df[3,]$filepath)
# All transcripts passing filter
findPeaksPerGene(cds, rfp, top_tx = 0)
# Top 50% of genes
findPeaksPerGene(cds, rfp)
```

`findUORFs` *Find upstream ORFs from transcript annotation*

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| `fiveUTRs` | (GRangesList) The 5' leaders or full transcript sequences |
| `fa` | a `FaFile`. With fasta sequences corresponding to fiveUTR annotation. Usually loaded from the genome of an organism with fa = ORFik:::findFa("path/to/fasta/genome") |
| `startCodon` | (character vector) Possible START codons to search for. Check `startDefinition` for helper function. Note that it is case sensitive, so "atg" would give 0 hits for a sequence with only capital "ATG" ORFs. |
| `stopCodon` | (character vector) Possible STOP codons to search for. Check `stopDefinition` for helper function. Note that it is case sensitive, so "tga" would give 0 hits for a sequence with only capital "TGA" ORFs. |
| `longestORF` | (logical) Default TRUE. Keep only the longest ORF per unique stopcodon: (seqname, strand, stopcodon) combination, Note: Not longest per transcript! You can also use function `longestORFs` after creation of ORFs for same result. |
| `minimumLength` | (integer) Default is 0. Which is START + STOP = 6 bp. Minimum length of ORF, without counting 3bps for START and STOP codons. For example minimumLength = 8 will result in size of ORFs to be at least START + 8*3 (bp) + STOP = 30 bases. Use this param to restrict search. |

cds (GRangesList) CDS of relative fiveUTRs, applicable only if you want to extend 5' leaders downstream of CDS's, to allow upstream ORFs that can overlap into CDS's.

cage Either a filePath for the CageSeq file as .bed .bam or .wig, with possible compressions (".gzip", ".gz", ".bgz"), or already loaded CageSeq peak data as GRanges or GAlignment. NOTE: If it is a .bam file, it will add a score column by running: convertToOneBasedRanges(cage, method = "5prime", addScoreColumn = TRUE) The score column is then number of replicates of read, if score column is something else, like read length, set the score column to NULL first.

extension The maximum number of basses upstream of the TSS to search for CageSeq peak.

filterValue The minimum number of reads on cage position, for it to be counted as possible new tss. (represented in score column in CageSeq data) If you already filtered, set it to 0.

restrictUpstreamToTx
a logical (FALSE). If TRUE: restrict leaders to not extend closer than 5 bases from closest upstream leader, set this to TRUE.

removeUnused logical (FALSE), if False: (standard is to set them to original annotation), If TRUE: remove leaders that did not have any cage support.

## Details

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

## Value

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

## See Also

Other findORFs: `findMapORFs()`, `findORFs()`, `findORFsFasta()`, `startDefinition()`, `stopDefinition()`

## Examples

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

```
## End(Not run)
```

---

`findUORFs_exp` *Find upstream ORFs from transcript annotation*

---

## Description

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

## Usage

```
findUORFs_exp(
  df,
  faFile = findFa(df),
  leaders = loadRegion(txdb, "leaders"),
  startCodon = startDefinition(1),
  stopCodon = stopDefinition(1),
  longestORF = TRUE,
  minimumLength = 0,
  overlappingCDS = FALSE,
  cage = NULL,
  extension = 1000,
  filterValue = 1,
  restrictUpstreamToTx = FALSE,
  removeUnused = FALSE,
  save_optimized = FALSE
)
```

## Arguments

| | |
|---|---|
| `df` | a txdb or `experiment` |
| `faFile` | FaFile of genome, default findFa(df). Default only works for ORFik experiments, if TxDb, input manually like: findFa(genome_path) |
| `leaders` | GRangesList, default: loadRegion(txdb, "leaders"). If you do not have any good leader annotation, a hack is to use `ORFik:::groupGRangesBy(startSites(loadRegion(txdb, "cds"), asGR = TRUE, keep.names = TRUE, is.sorted = TRUE))` |
| `startCodon` | (character vector) Possible START codons to search for. Check `startDefinition` for helper function. Note that it is case sensitive, so "atg" would give 0 hits for a sequence with only capital "ATG" ORFs. |
| `stopCodon` | (character vector) Possible STOP codons to search for. Check `stopDefinition` for helper function. Note that it is case sensitive, so "tga" would give 0 hits for a sequence with only capital "TGA" ORFs. |

| | |
|---|---|
| longestORF | (logical) Default TRUE. Keep only the longest ORF per unique stopcodon: (seqname, strand, stopcodon) combination, Note: Not longest per transcript! You can also use function longestORFs after creation of ORFs for same result. |
| minimumLength | (integer) Default is 0. Which is START + STOP = 6 bp. Minimum length of ORF, without counting 3bps 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. |
| overlappingCDS | logical, default FALSE. Include uORFs that overlap CDS. |
| cage | Either a filePath for the CageSeq file as .bed .bam or .wig, with possible compressions (".gzip", ".gz", ".bgz"), or already loaded CageSeq peak data as GRanges or GAlignment. NOTE: If it is a .bam file, it will add a score column by running: convertToOneBasedRanges(cage, method = "5prime", addScoreColumn = TRUE) The score column is then number of replicates of read, if score column is something else, like read length, set the score column to NULL first. |
| extension | The maximum number of basses upstream of the TSS to search for CageSeq peak. |
| filterValue | The minimum number of reads on cage position, for it to be counted as possible new tss. (represented in score column in CageSeq data) If you already filtered, set it to 0. |
| restrictUpstreamToTx | a logical (FALSE). If TRUE: restrict leaders to not extend closer than 5 bases from closest upstream leader, set this to TRUE. |
| removeUnused | logical (FALSE), if False: (standard is to set them to original annotation), If TRUE: remove leaders that did not have any cage support. |
| save_optimized | logical, default FALSE. If TRUE, save in the optimized folder for the experiment. You must have made this directory before running this function (call makeTxdbFromGenome first if not). |

## Details

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

## Value

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

## See Also

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

## Examples

```
df <- ORFik.template.experiment()
# Without cds overlapping, no 5' leader extension
findUORFs_exp(df, extension = 0)
```

```
# Without cds overlapping, extends 5' leaders by 1000 (good for yeast etc)
findUORFs_exp(df)
# Include cds overlapping uorfs
findUORFs_exp(df, overlappingCDS = TRUE)
```

---

`find_url_ebi` *Locates and check if fastq files exists in ebi*

---

## Description

Look for files in ebi following url: ftp://ftp.sra.ebi.ac.uk/vol1/fastq Paired end and single end fastq files.
EBI uses 3 ways to organize data inside vol1/fastq:
- 1: Most common: SRR(3 first)/0(2 last)/whole
- 2: less common: SRR(3 first)/00(1 last)/whole
- 3: least common SRR(3 first)/whole

## Usage

```
find_url_ebi(SRR, stop.on.error = FALSE, study = NULL)
```

## Arguments

| | |
|---|---|
| `SRR` | character, SRR, ERR or DRR numbers. |
| `stop.on.error` | logical FALSE, if TRUE will stop if all files are not found. If FALSE returns empty character vector if error is catched. |
| `study` | default NULL, optional PRJ (study id) to speed up search for URLs. |

## Value

full url to fastq files, same length as input (2 urls for paired end data). Returns empty character() if all files not found.

## Examples

```
# Test the 3 ways to get fastq files from EBI
# Both single end and paired end data

# Most common: SRR(3 first)/0(2 last)/whole
# Single
ORFik:::find_url_ebi("SRR10503056")
# Paired
ORFik:::find_url_ebi("SRR10500056")

# less common: SRR(3 first)/00(1 last)/whole
# Single
#ORFik:::find_url_ebi("SRR1562873")
# Paired
```

```
#ORFik:::find_url_ebi("SRR1560083")
# least common SRR(3 first)/whole
# Single
#ORFik:::find_url_ebi("SRR105687")
# Paired
#ORFik:::find_url_ebi("SRR105788")
```

---

`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, default: (TRUE) |

## 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, default: (TRUE) |

## 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)
```

---

# fix_malformed_gff *Fix a malformed gff file*

---

## Description

Basically removes all info lines with character length > 32768 and save that new file.

## Usage

```
fix_malformed_gff(gff)
```

## Arguments

| | |
|---|---|
| gff | character, path to gtf, can not be gzipped! |

## Value

path of fixed gtf

## Examples

```
# fix_malformed_gff("my_bad_gff.gff")
```

## flankPerGroup *Get flanks per group*

### Description

For a GRangesList, get start and end site, return back as GRL.

### Usage

```
flankPerGroup(grl)
```

### Arguments

| | |
|---|---|
| grl | a GRangesList |

### Value

a GRangesList, 1 GRanges per group with: start as minimum start of group and end as maximum per group.

### Examples

```
grl <- GRangesList(tx1 = GRanges("1", IRanges(c(1,5), width = 2), "+"),
                   tx2 = GRanges("2", IRanges(c(10,15), width = 2), "+"))
flankPerGroup(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 read widths) that overlap ORFs and normalize by CDS read width fractions. So if all read length are width 34 in ORFs and CDS, value is 1. If width is 33 in ORFs and 34 in CDS, value is 0. If width is 33 in ORFs and 50/50 (33 and 34) in CDS, values will be 0.5 (for 33).

### Usage

```
floss(grl, RFP, cds, start = 26, end = 34, weight = 1L)
```

## Arguments

| | |
|---|---|
| grl | a GRangesList object can be either transcripts, 5' utrs, cds', 3' utrs or ORFs as a special case (uORFs, potential new cds' etc). If regions are not spliced you can send a GRanges object. |
| RFP | ribosomal footprints, given as GAlignments or GRanges object, must be already shifted and resized to the p-site. Requires a $size column with original read lengths. |
| 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 (inclusive) |
| end | usually 34, the end of the floss interval (inclusive) |
| weight | a vector (default: 1L, if 1L it is identical to countOverlaps()), if single number (!= 1), it applies for all, if more than one must be equal size of 'reads'. else it must be the string name of a defined meta column in subject "reads", that gives number of times a read was found. GRanges("chr1", 1, "+", score = 5), would mean "score" column tells that this alignment region was found 5 times. |

## Details

Pseudo explanation of the function:

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

Where 'name' is transcript names.
Please read more in the article.

## Value

a vector of FLOSS of length same as grl, 0 means no RFP reads in range, 1 is perfect match.

## References

doi: 10.1016/j.celrep.2014.07.045

## See Also

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

## Examples

```
ORF1 <- GRanges(seqnames = "1",
                ranges = IRanges(start = c(1, 12, 22),
                end = c(10, 20, 32)),
                strand = "+")
```

```
grl <- GRangesList(tx1_1 = ORF1)
# RFP is 1 width position based GRanges
RFP <- GRanges("1", IRanges(c(1, 25, 35, 38), width = 1), "+")
RFP$size <- c(28, 28, 28, 29) # original width in size 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)

# With repeated alignments in score column
ORF2 <- GRanges(seqnames = "1",
                ranges = IRanges(start = c(12, 22, 36),
                end = c(20, 32, 38)),
                strand = "+")
grl <- GRangesList(tx1_1 = ORF1, tx1_2 = ORF2)
score(RFP) <- c(5, 10, 5, 10)
floss(grl, RFP, cds, weight = "score")
```

---

`fpkm` *Create normalizations of overlapping read counts.*

---

## Description

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

## Usage

```
fpkm(grl, reads, pseudoCount = 0, librarySize = "full", weight = 1L)
```

## Arguments

| | |
|---|---|
| `grl` | a `GRangesList` object can be either transcripts, 5' utrs, cds', 3' utrs or ORFs as a special case (uORFs, potential new cds' etc). If regions are not spliced you can send a `GRanges` object. |
| `reads` | a `GAlignments`, `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. |
| `librarySize` | either numeric value or character vector. Default ("full"), number of alignments in library (reads). If you just have a subset, you can give the value by librarySize = length(wholeLib), if you want lib size to be only number of reads overlapping grl, do: librarySize = "overlapping" sum(countOverlaps(reads, grl) > 0), if reads[1] has 3 hits in grl, and reads[2] has 2 hits, librarySize will be 2, not 5. You can also get the inverse overlap, if you want lib size to be total number of overlaps, do: librarySize = "DESeq" This is standard fpkm way of |

DESeq2::fpkm(robust = FALSE) sum(countOverlaps(grl, reads)) if grl[1] has 3 reads and grl[2] has 2 reads, librarySize is 5, not 2.

`weight` a vector (default: 1L, if 1L it is identical to countOverlaps()), if single number (!= 1), it applies for all, if more than one must be equal size of 'reads'. else it must be the string name of a defined meta column in subject "reads", that gives number of times a read was found. GRanges("chr1", 1, "+", score = 5), would mean "score" column tells that this alignment region was found 5 times.

## Details

Note also that you must consider if you will use the whole read library or just the reads overlapping 'grl' for library size. A normal question here is, does it make sense to include rRNA in library size ? If you only want overlapping grl, do: librarySize = "overlapping"

## Value

a numeric vector with the fpkm values

## References

doi: 10.1038/nbt.1621

## See Also

Other features: `computeFeatures()`, `computeFeaturesCage()`, `countOverlapsW()`, `disengagementScore()`, `distToCds()`, `distToTSS()`, `entropy()`, `floss()`, `fpkm_calc()`, `fractionLength()`, `initiationScore()`, `insideOutsideORF()`, `isInFrame()`, `isOverlapping()`, `kozakSequenceScore()`, `orfScore()`, `rankOrder()`, `ribosomeReleaseScore()`, `ribosomeStallingScore()`, `startRegion()`, `startRegionCoverage()`, `stopRegion()`, `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)

# With weights (10 reads at position 25)
RFP <- GRanges("1", IRanges(25, 25),"+", score = 10)
fpkm(grl, RFP, weight = "score")
```

`fractionLength` *Fraction Length*

## Description

Fraction Length is defined as

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

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

## Usage

```
fractionLength(grl, tx_len = widthPerGroup(tx, TRUE), tx = NULL)
```

## 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)'.<br>If you used CageSeq to reannotate leaders, then the tss for the the leaders have changed, therefore the tx lengths have changed. To account for that call: 'tx_len <- widthPerGroup(extendLeaders(tx, cageFiveUTRs))' and calculate fraction length using 'fractionLength(grl, tx_len)'. |
| `tx` | default NULL, a `GRangesList` object of transcript to get lengths from. Pass in for wrapping to widths inside the function. |

## Value

a numeric vector of ratios

## References

doi: 10.1242/dev.098343

## See Also

Other features: `computeFeatures()`, `computeFeaturesCage()`, `countOverlapsW()`, `disengagementScore()`, `distToCds()`, `distToTSS()`, `entropy()`, `floss()`, `fpkm()`, `fpkm_calc()`, `initiationScore()`, `insideOutsideORF()`, `isInFrame()`, `isOverlapping()`, `kozakSequenceScore()`, `orfScore()`, `rankOrder()`, `ribosomeReleaseScore()`, `ribosomeStallingScore()`, `startRegion()`, `startRegionCoverage()`, `stopRegion()`, `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, tx = tx)
```

---

`fread.bed` *Load bed file as GRanges*

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| `filePath` | The location of the bed file |
| `chrStyle` | a GRanges object, TxDb, FaFile, , a `seqlevelsStyle` or `Seqinfo`. (Default: NULL) to get seqlevelsStyle from. In addition if it is a Seqinfo object, seqinfo will be updated. Example of seqlevelsStyle update: Is chromosome 1 called chr1 or 1, is mitocondrial chromosome called MT or chrM etc. Will use 1st seqlevel-style if more are present. Like: c("NCBI", "UCSC") -> pick "NCBI" |

## Value

a `GRanges` object

## See Also

Other utils: `bedToGR()`, `convertToOneBasedRanges()`, `export.bed12()`, `export.bigWig()`, `export.fstwig()`, `export.wiggle()`, `fimport()`, `findFa()`, `optimizeReads()`, `readBam()`, `readBigWig()`, `readWig()`

## Examples

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

gcContent *Get GC content*

## Description

0.5 means 50

## Usage

```
gcContent(seqs, fa = NULL)
```

## Arguments

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

## Value

a numeric vector of gc content scores

## Examples

```
# 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)
# get path to FaFile for sequences
faFile <- system.file("extdata/references/danio_rerio", "genome_dummy.fasta",
 package = "ORFik")
gcContent(ORFs, faFile)
```

geneToSymbol *Get gene symbols from Ensembl gene ids*

## Description

If your organism is not in this list of supported organisms, manually assign the input arguments. There are 2 main fetch modes:
By gene ids (Single accession per gene)
By tx ids (Multiple accessions per gene)
Run the mode you need depending on your required attributes.

Will check for already existing table of all genes, and use that instead of re-downloading every time (If you input valid experiment or txdb and have run `makeTxdbFromGenome` with symbols = TRUE, you have a file called gene_symbol_tx_table.fst) will load instantly. If df = NULL, it can still search cache to load a bit slower.

## Usage

```
geneToSymbol(
  df,
  organism_name = organism(df),
  gene_ids = filterTranscripts(df, by = "gene", 0, 0, 0),
 org.dataset = paste0(tolower(substr(organism_name, 1, 1)), gsub(".* ", replacement =
    "", organism_name), "_gene_ensembl"),
  ensembl = biomaRt::useEnsembl("ensembl", dataset = org.dataset),
  attribute = "external_gene_name",
  include_tx_ids = FALSE,
  uniprot_id = FALSE,
  force = FALSE,
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| df | an ORFik `experiment` or TxDb object with defined organism slot. If set will look for file at path of txdb / experiment reference path named: 'gene_symbol_tx_table.fst' relative to the txdb/genome directory. Can be set to NULL if gene_ids and organism is defined manually. |
| organism_name | default, `organism(df)`. Scientific name of organism, like ("Homo sapiens"), remember capital letter for first name only! |
| gene_ids | default, `filterTranscripts(df, by = "gene", 0, 0, 0)`. Ensembl gene IDs to search for (default all transcripts coding and noncoding) To only get coding do: filterTranscripts(df, by = "gene", 0, 1, 0) |
| org.dataset | default, `paste0(tolower(substr(organism_name, 1, 1)), gsub(".* ", replacement = "", organism_name), "_gene_ensembl")` the ensembl dataset to use. For Homo sapiens, this converts to default as: hsapiens_gene_ensembl |
| ensembl | default, `useEnsembl("ensembl",dataset=org.dataset)` .The mart connection. |
| attribute | default, "external_gene_name", the biomaRt column / columns default(primary gene symbol names). These are always from specific database, like hgnc symbol for human, and mgi symbol for mouse and rat, sgd for yeast etc. |

include_tx_ids — logical, default FALSE, also match tx ids, which then returns as the 3rd column. Only allowed when 'df' is defined. If

uniprot_id — logical, default FALSE. Include uniprotsptrembl and/or uniprotswissprot. If include_tx_ids you will get per isoform if available, else you get canonical uniprot id per gene. If both uniprotsptrembl and uniprotswissprot exists, it will make a merged uniprot id column with rule: if id exists in uniprotswissprot, keep. If not, use uniprotsptrembl column id.

force — logical FALSE, if TRUE will not look for existing file made through makeTxdbFromGenome corresponding to this txdb / ORFik experiment stored with name "gene_symbol_tx_table.fst".

verbose — logical TRUE, if FALSE, do not output messages.

## Value

data.table with 2, 3 or 4 columns: gene_id, gene_symbol, tx_id and uniprot_id named after attribute, sorted in order of gene_ids input. (example: returns 3 columns if include_tx_ids is TRUE), and more if additional columns are specified in 'attribute' argument.

## Examples

```
## Without ORFik experiment input
gene_id_ATF4 <- "ENSG00000128272"
#geneToSymbol(NULL, organism_name = "Homo sapiens", gene_ids = gene_id_ATF4)
# With uniprot canonical isoform id:
#geneToSymbol(NULL, organism_name = "Homo sapiens", gene_ids = gene_id_ATF4, uniprot_id = TRUE)

## All genes from Organism using ORFik experiment
# df <- read.experiment("some_experiment)
# geneToSymbol(df)

## Non vertebrate species (the ones not in ensembl, but in ensemblGenomes mart)
#txdb_ylipolytica <- loadTxdb("txdb_path")
#dt2 <- geneToSymbol(txdb_ylipolytica, include_tx_ids = TRUE,
#   ensembl = useEnsemblGenomes(biomart = "fungi_mart", dataset = "ylipolytica_eg_gene"))
```

---

# getGenomeAndAnnotation

*Download genome (fasta), annotation (GTF) and contaminants*

---

## Description

This function automatically downloads (if files not already exists) genomes and contaminants specified for genome alignment. By default, it will use ensembl reference, upon completion, the function will store a file called `file.path(output.dir, "outputs.rds")` with the output paths of your completed genome/annotation downloads. For most non-model nonvertebrate organisms, you need my fork of biomartr for it to work: remotes::install_github("Roleren/biomartr) If you misspelled something or crashed, delete wrong files and run again.
Do remake = TRUE, to do it all over again.

## Usage

```
getGenomeAndAnnotation(
  organism,
  output.dir,
  db = "ensembl",
  GTF = TRUE,
  genome = TRUE,
  merge_contaminants = TRUE,
  phix = FALSE,
  ncRNA = FALSE,
  tRNA = FALSE,
  rRNA = FALSE,
  gunzip = TRUE,
  remake = FALSE,
  assembly_type = c("primary_assembly", "toplevel"),
  optimize = FALSE,
  gene_symbols = FALSE,
  uniprot_id = FALSE,
  pseudo_5UTRS_if_needed = NULL,
  remove_annotation_outliers = TRUE,
  notify_load_existing = TRUE,
  assembly = organism
)
```

## Arguments

| | |
|---|---|
| organism | scientific name of organism, Homo sapiens, Danio rerio, Mus musculus, etc. See `biomartr:::get.ensembl.info()` for full list of supported organisms. |
| output.dir | directory to save downloaded data |
| db | database to use for genome and GTF, default adviced: "ensembl" (remember to set assembly_type to "primary_assembly", else it will contain haplotypes, very large file!). Alternatives: "refseq" (reference assemblies) and "genbank" (all assemblies) |
| GTF | logical, default: TRUE, download gtf of organism specified in "organism" argument. If FALSE, check if the downloaded file already exist. If you want to use a custom gtf from you hard drive, set GTF = FALSE, and assign:<br>annotation <- getGenomeAndAnnotation(gtf = FALSE)<br>annotation["gtf"] = "path/to/gtf.gtf".<br>If db is not "ensembl", you will instead get a gff file. |
| genome | logical, default: TRUE, download genome of organism specified in "organism" argument. If FALSE, check if the downloaded file already exist. If you want to use a custom gtf from you hard drive, set `GTF = FALSE`, and assign:<br>`annotation <- getGenomeAndAnnotation(genome = FALSE)`<br>`annotation["genome"] = "path/to/genome.fasta"`.<br>Will download the primary assembly from Ensembl. |
| merge_contaminants | logical, default TRUE. Will merge the contaminants specified into one fasta file, |

this considerably saves space and is much quicker to align with STAR than each contaminant on it's own. If no contaminants are specified, this is ignored.

`phix` logical, default FALSE, download phiX sequence to filter out Illumina control reads. ORFik defines Phix as a contaminant genome. Phix is used in Illumina sequencers for sequencing quality control. Genome is: refseq, Escherichia phage phiX174. If sequencing facility created fastq files with the command `bcl2fastq`, then there should be very few phix reads left in the fastq files recieved.

`ncRNA` logical or character, default FALSE (not used, no download), if TRUE or defned path, ncRNA is used as a contaminant reference. If TRUE, will try to find ncRNA sequences from the gtf file, usually represented as lncRNA (long non-coding RNA's). Will let you know if no ncRNA sequences were found in gtf.
If not found try character input:
Alternatives; "auto": Will try to find ncRNA file on NONCODE from organism, Homo sapiens -> human etc. "auto" will not work for all, then you must specify the name used by NONCODE, go to the link below and find it. If not "auto" / "" it must be a character vector of species common name (not scientific name) Homo sapiens is human, Rattus norwegicus is rat etc, download ncRNA sequence to filter out with. From NONCODE online server, if you cant find common name see: http://www.noncode.org/download.php/

`tRNA` logical or character, default FALSE (not used, no download), tRNA is used as a contaminant genome. If TRUE, will try to find tRNA sequences from the gtf file, usually represented as Mt_tRNA (mature tRNA's). Will let you know if no tRNA sequences were found in gtf. If not found try character input:
if not "" it must be a character vector to valid path of mature tRNAs fasta file to remove as contaminants on your disc. Find and download your wanted mtRNA at: http://gtrnadb.ucsc.edu/, or run trna-scan on you genome.

`rRNA` logical or character, default FALSE (not used, no download), rRNA is used as a contaminant reference If TRUE, will try to find rRNA sequences from the gtf file, usually represented as rRNA (ribosomal RNA's). Will let you know if no rRNA sequences were found in gtf. If not found you can try character input:
If "silva" will download silva SSU & LSU sequences for all species (250MB file) and use that. If you want a smaller file go to https://www.arb-silva.de/
If not "" or "silva" it must be a character vector to valid path of mature rRNA fasta file to remove as contaminants on your disc.

`gunzip` logical, default TRUE, uncompress downloaded files that are zipped when downloaded, should be TRUE!

`remake` logical, default: FALSE, if TRUE remake everything specified

`assembly_type` character, default c("primary_assembly", "toplevel"). Used for ensembl only, specifies the genome assembly type. Searches for both primary and toplevel, and if both are found, uses the first by order (so primary is prioritized by default). The Primary assembly should usually be used if it exists. The "primary assembly" contains all the top-level sequence regions, excluding alternative haplotypes and patches. If the primary assembly file is not present for a species (only defined for standard model organisms), that indicates that there were no haplotype/patch regions, and in such cases, the 'toplevel file is used. For more details see: ensembl tutorial

`optimize` logical, default FALSE. Create a folder within the folder of the gtf, that includes optimized objects to speed up loading of annotation regions from up to 15 seconds on human genome down to 0.1 second. ORFik will then load these optimized objects instead. Currently optimizes filterTranscript() function and loadRegion() function for 5' UTRs, 3' UTRs, CDS, mRNA (all transcript with CDS) and tx (all transcripts).

`gene_symbols` logical default FALSE. If TRUE, will download and store all gene symbols for all transcripts (coding and noncoding)- In a file called: "gene_symbol_tx_table.fst" in same folder as txdb. hgcn for human, mouse symbols for mouse and rat, more to be added.

`uniprot_id` logical default FALSE. If TRUE, will download and store all uniprot id for all transcripts (coding and noncoding)- In a file called: "gene_symbol_tx_table.fst" in same folder as txdb.

`pseudo_5UTRS_if_needed`
integer, default NULL. If defined > 0, will add pseudo 5' UTRs if 30 a leader.

`remove_annotation_outliers`
logical, default TRUE. Only for refseq. shall outlier lines be removed from the input annotation_file? If yes, then the initial annotation_file will be overwritten and the removed outlier lines will be stored at tempdir for further exploration. Among others Aridopsis refseq contains malformed lines, where this is needed

`notify_load_existing`
logical, default TRUE. If annotation exists (defined as: locally (a file called outputs.rds) exists in outputdir), print a small message notifying the user it is not redownloading. Set to FALSE, if this is not wanted

`assembly` character, default is assembly = organism, which means getting the first assembly in list, otherwise the name of the assembly wanted, like "GCA_000005845" will get ecoli substrain k12, which is the most used ones for references. Usually ignore this for non bacterial species.

## Details

Some files that are made after download:
- A fasta index for the genome
- A TxDb to speed up GTF/GFF reading
- Seperat of merged contaminant files
Files that can be made:
- Gene symbols (hgnc, etc)
- Uniprot ids (For name of protein structures)
If you want custom genome or gtf from you hard drive, assign existing paths like this:
annotation <- getGenomeAndAnnotation(GTF = "path/to/gtf.gtf", genome = "path/to/genome.fasta")

## Value

a named character vector of path to genomes and gtf downloaded, and additional contaminants if used. If merge_contaminants is TRUE, will not give individual fasta files to contaminants, but only the merged one.

## See Also

Other STAR: STAR.align.folder(), STAR.align.single(), STAR.allsteps.multiQC(), STAR.index(), STAR.install(), STAR.multiQC(), STAR.remove.crashed.genome(), install.fastp()

## Examples

```
## Get Saccharomyces cerevisiae genome and gtf (create txdb for R)
#getGenomeAndAnnotation("Saccharomyces cerevisiae", tempdir(), assembly_type = "toplevel")
## Download and add pseudo 5' UTRs
#getGenomeAndAnnotation("Saccharomyces cerevisiae", tempdir(), assembly_type = "toplevel",
#  pseudo_5UTRS_if_needed = 100)
## Get Danio rerio genome and gtf (create txdb for R)
#getGenomeAndAnnotation("Danio rerio", tempdir())

output.dir <- "/Bio_data/references/zebrafish"
## Get Danio rerio and Phix contamints to deplete during alignment
#getGenomeAndAnnotation("Danio rerio", output.dir, phix = TRUE)

## Optimize for ORFik (speed up for large annotations like human or zebrafish)
#getGenomeAndAnnotation("Danio rerio", tempdir(), optimize = TRUE)

# Drosophila melanogaster (toplevel exists only)
#getGenomeAndAnnotation("drosophila melanogaster", output.dir = file.path(config["ref"],
# "Drosophila_melanogaster_BDGP6"), assembly_type = "toplevel")
## How to save malformed refseq gffs:
## First run function and let it crash:
#annotation <- getGenomeAndAnnotation(organism = "Arabidopsis thaliana",
#  output.dir = "~/Desktop/test_plant/",
#  assembly_type = "primary_assembly", db = "refseq")
## Then apply a fix (example for linux, too long rows):
# fixed_gff <- fix_malformed_gff("~/Desktop/test_plant/Arabidopsis_thaliana_genomic_refseq.gff")
## Then updated arguments:
# annotation <- c(fixed_gff, "~/Desktop/test_plant/Arabidopsis_thaliana_genomic_refseq.fna")
# names(annotation) <- c("gtf", "genome")
# Then make the txdb (for faster R use)
# makeTxdbFromGenome(annotation["gtf"], annotation["genome"], organism = "Arabidopsis thaliana")
```

---

# get_bioproject_candidates

*Query eutils for bioproject IDs*

---

## Description

The default query of Ribosome Profiling human, will result in internal entrez search of: Ribosome[All Fields] AND Profiling[All Fields] AND ("Homo sapiens"[Organism] OR human[All Fields])

### Usage

```
get_bioproject_candidates(
  term = "Ribosome Profiling human",
  as_accession = TRUE,
  add_study_title = FALSE,
  RetMax = 10000
)
```

### Arguments

| | |
|---|---|
| term | character, default "Ribosome Profiling human". A space is translated into AND, that means "Ribosome AND Profiling AND human", will give same as above. To do OR operation, do: "Ribosome OR profiling OR human". |
| as_accession | logical, default TRUE. Get bioproject accessions: PRJNA, PRJEB, PRJDB values, or IDs (FALSE), numbers only. Accessions are usually the thing needed for most tools. |
| add_study_title | logical, default FALSE. If TRUE, return as data table with 2 columns: id: ID or accessions. title: The title of the study. |
| RetMax | integer, default 10000. How many IDs to return maximum |

### Value

character vector of Accessions or IDs. If add_study_title is TRUE, returns a data.table.

### References

https://www.ncbi.nlm.nih.gov/books/NBK25501/

### See Also

Other sra: browseSRA(), download.SRA(), download.SRA.metadata(), download.ebi(), install.sratoolkit(), rename.SRA.files()

### Examples

```
term <- "Ribosome Profiling Saccharomyces cerevisiae"
# get_bioproject_candidates(term)
```

---

## get_silva_rRNA *Download Silva SSU & LSU sequences*

---

### Description

Version downloaded is 138.1. NR99_tax (non redundant)

### Usage

```
get_silva_rRNA(output.dir)
```

### Arguments

| | |
|---|---|
| output.dir | directory to save downloaded data |

### Details

If it fails from timeout, set higher timeout: options(timeout = 200)

### Value

filepath to downloaded file

### Examples

```
output.dir <- tempdir()
# get_silva_rRNA(output.dir)
```

---

## 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.
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))‘.

### Usage

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

### Arguments

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

### Details

It is important that all intended groups in ‘other‘ are uniquely named, otherwise duplicated group names will be grouped together.

### Value

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

## Examples

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

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

---

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

---

## Description

Get number of ranges per group as an iteration

## Usage

```
groupings(grl)
```

## Arguments

| | |
|---|---|
| `grl` | GRangesList |

## Value

an integer vector

## Examples

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

heatMapRegion *Create coverage heatmaps of specified region*

## Description

Simplified input space for easier abstraction of coverage heatmaps
Pick your transcript region and plot directly
Input CAGE file if you use TSS and want improved 5' annotation.

## Usage

```
heatMapRegion(
  df,
  region = "TIS",
  outdir = "default",
  scores = c("transcriptNormalized", "sum"),
  type = "ofst",
  cage = NULL,
  plot.ext = ".pdf",
  acceptedLengths = 21:75,
  upstream = c(50, 30),
  downstream = c(29, 69),
  shifting = c("5prime", "3prime"),
  longestPerGene = TRUE,
  colors = "default",
  scale_x = 5.5,
  scale_y = 15.5,
  gradient.max = "default",
  BPPARAM = BiocParallel::SerialParam()
)
```

## Arguments

| | |
|---|---|
| df | an ORFik experiment |
| region | a character, default "TIS". The centering point for the heatmap (what is position 0, beween -50 and 50 etc), can be any combination of the set: c("TSS", "TIS", "TTS", "TES"), which are: - Transcription start site (5' end of mrna)<br>- Translation initation site (5' end of CDS)<br>- Translation termination site (5' end of 3' UTRs)<br>- Transcription end site (3' end of 3' UTRs) |
| outdir | a character path, default: "default", saves to: `file.path(QCfolder(df), "heatmaps/")`, a created folder within the ORFik experiment data folder for plots. Change if you want custom location. |
| scores | character vector, default `c("transcriptNormalized", "sum")`, either of zscore, transcriptNormalized, sum, mean, median, .. see ?coverageScorings for info and more alternatives. |

| | |
|---|---|
| type | character, default: "ofst". Type of library: either "default", usually bam format (the one you gave to experiment), "pshifted" pshifted reads, "ofst", "bed", "bedo" optimized bed, or "wig" |
| cage | a character path to library file or a GRanges, GAlignments preloaded file of CAGE data. Only used if "TSS" is defined as region, to redefine 5' leaders. |
| plot.ext | a character, default ".pdf", alternative ".png" |
| acceptedLengths | an integer vector (NULL), the read lengths accepted. Default NULL, means all lengths accepted. |
| upstream | 1 or 2 integers, default c(50, 30), how long upstream from 0 should window extend (first index is 5' end extension, second is 3' end extension). If only 1 shifting, only 1 value should be given, if two are given will use first. |
| downstream | 1 or 2 integers, default c(29, 69), how long upstream from 0 should window extend (first index is 5' end extension, second is 3' end extension). If only 1 shifting, only 1 value should be given, if two are given will use first. |
| shifting | a character, default c("5prime", "3prime"), can also be NULL (no shifting of reads). If NULL, will use first index of 'upstream' and 'downstream' argument. |
| longestPerGene | logical, default TRUE. Use only longest transcript isoform per gene. This will speed up your computation. |
| colors | character vector, default: "default", this gives you: c("white", "yellow2", "yellow3", "lightblue", "blue", "navy"), do "high" for more high contrasts, or specify your own colors. |
| scale_x | numeric, how should the width of the single plots be scaled, bigger the number, the bigger the plot |
| scale_y | numeric, how should the height of the plots be scaled, bigger the number, the bigger the plot |
| gradient.max | numeric or character, default: "default", which is: max(coverage$score), the max coverage over all readlengths. If you want all plots to use same reference point for max scaling, then first detect this point, look at max in plot etc, and use that value, to get all plots to have same max point. |
| BPPARAM | a core param, default: single thread: BiocParallel::SerialParam(). Set to BiocParallel::bpparam() to use multicore. Be aware, this uses a lot of extra ram (40GB+) for larger human samples! |

## Value

invisible(NULL), plots are saved

## See Also

Other heatmaps: coverageHeatMap(), heatMapL(), heatMap_single()

## Examples

```
# Toy example, will not give logical output, but shows how it works
df <- ORFik.template.experiment()[9:10,] # Subset to 2 Ribo-seq libs
#heatMapRegion(df, "TIS", outdir = "default")
#
# Do also TSS, add cage for specific TSS
# heatMapRegion(df, c("TSS", "TIS"), cage = "path/to/cage.bed")

# Do on pshifted reads instead of original files
remove.experiments(df) # Remove loaded experiment first
# heatMapRegion(df, "TIS", type = "pshifted")
```

---

# heatMap_single *Coverage heatmap of single libraries*

---

## Description

Coverage heatmap of single libraries

## Usage

```
heatMap_single(
  region,
  tx,
  reads,
  outdir,
  scores = "sum",
  upstream,
  downstream,
  zeroPosition = upstream,
  returnCoverage = FALSE,
  acceptedLengths = NULL,
  legendPos = "right",
  colors = "default",
  addFracPlot = TRUE,
  location = "start site",
  shifting = NULL,
  skip.last = FALSE,
  title = NULL,
  gradient.max = "default"
)
```

## Arguments

| | |
|---|---|
| region | #' a GRangesList object of region, usually either leaders, cds', 3' utrs or ORFs, start region, stop regions etc. This is the region that will be mapped in heatmap |

| | |
|---|---|
| tx | default NULL, a GRangesList of transcripts or (container region), names of tx must contain all grl names. The names of grl can also be the ORFik orf names. that is "txName_id" |
| reads | a GAlignments, GRanges, or precomputed coverage as covRleList (where names of covRle objects are readlengths) of RiboSeq, RnaSeq etc.<br>Weigths for scoring is default the 'score' column in 'reads'. Can also be random access paths to bigWig or fstwig file. Do not use random access for more than a few genes, then loading the entire files is usually better. |
| outdir | a character path to save file as: not just directory, but full name. |
| scores | character vector, default "sum", either of zscore, transcriptNormalized, sum, mean, median, .. see ?coverageScorings for info and more alternatives. |
| upstream | an integer, relative region to get upstream from. |
| downstream | an integer, relative region to get downstream from |
| zeroPosition | an integer DEFAULT (upstream), what is the center point? Like leaders and cds combination, then 0 is the TIS and -1 is last base in leader. NOTE!: if windows have different widths, this will be ignored. |
| returnCoverage | logical, default: FALSE, return coverage, if FALSE returns plot instead. |
| acceptedLengths | an integer vector (NULL), the read lengths accepted. Default NULL, means all lengths accepted. |
| legendPos | a character, Default "right". Where should the fill legend be ? ("top", "bottom", "right", "left") |
| colors | character vector, default: "default", this gives you: c("white", "yellow2", "yellow3", "lightblue", "blue", "navy"), do "high" for more high contrasts, or specify your own colors. |
| addFracPlot | Add margin histogram plot on top of heatmap with fractions per positions |
| location | a character, default "start site", will make xlabel of heatmap be Position relative to "start site" or alternative given. |
| shifting | a character, default NULL (no shifting), can also be either of c("5prime", "3prime") |
| skip.last | skip top(highest) read length, default FALSE |
| title | a character, default NULL (no title), what is the top title of plot? |
| gradient.max | numeric or character, default: "default", which is: max(coverage$score), the max coverage over all readlengths. If you want all plots to use same reference point for max scaling, then first detect this point, look at max in plot etc, and use that value, to get all plots to have same max point. |

## Value

ggplot2 grob (default), data.table (if returnCoverage is TRUE)

## See Also

Other heatmaps: coverageHeatMap(), heatMapL(), heatMapRegion()

`import.bedo` *Load GRanges object from .bedo*

## Description

.bedo is .bed ORFik, an optimized bed format for coverage reads with read lengths .bedo is a text based format with columns (6 maximum):
1. chromosome
2. start
3. end
4. strand
5. ref width (cigar # M's, match/mismatch total)
6. duplicates of that read

## Usage

```
import.bedo(path)
```

## Arguments

| | |
|---|---|
| `path` | a character, location on disc (full path) |

## Details

Positions are 1-based, not 0-based as .bed. export with export.bedo

## Value

GRanges object

`import.bedoc` *Load GAlignments object from .bedoc*

## Description

A much faster way to store, load and use bam files.
.bedoc is .bed ORFik, an optimized bed format for coverage reads with cigar and replicate number.
.bedoc is a text based format with columns (5 maximum):
1. chromosome
2. cigar: (cigar # M's, match/mismatch total)
3. start (left most position)
4. strand (+, -, *)
5. score: duplicates of that read

### Usage

```
import.bedoc(path)
```

### Arguments

| | |
|---|---|
| path | a character, location on disc (full path) |

### Details

Positions are 1-based, not 0-based as .bed. export with export.bedo

### Value

GAlignments object

---

## import.fstwig *Import region from fastwig*

---

### Description

Import region from fastwig

### Usage

```
import.fstwig(gr, dir, id = "", readlengths = "all")
```

### Arguments

| | |
|---|---|
| gr | a GRanges object of exons |
| dir | prefix to filepath for file strand and chromosome will be added |
| id | id to column type, not used currently! |
| readlengths | integer / character vector, default "all". Or a subset of readlengths. |

### Value

a data.table with columns specified by readlengths

`import.ofst` *Load GRanges / GAlignments object from .ofst*

## Description

A much faster way to store, load and use bam files.
.ofst is ORFik fast serialized object, an optimized format for coverage reads with cigar and replicate number. It uses the fst format as back-end: `fst-package`.
A .ofst ribo seq file can compress the information in a bam file from 5GB down to a few MB. This new files has super fast reading time, only a few seconds, instead of minutes. It also has random index access possibility of the file.
.ofst is represented as a data.frane format with minimum 4 columns:
1. chromosome
2. start (left most position)
3. strand (+, -, *)
4. width (not added if cigar exists)
5. cigar (not needed if width exists): (cigar # M's, match/mismatch total)
5. score: duplicates of that read
6. size: qwidth according to reference of read

If file is from `GAlignmentPairs`, it will contain a cigar1, cigar2 instead of cigar and start1 and start2 instead of start

## Usage

```
import.ofst(file, strandMode = 0, seqinfo = NULL)
```

## Arguments

| | |
|---|---|
| `file` | a path to a .ofst file |
| `strandMode` | numeric, default 0. Only used for paired end bam files. One of (0: strand = *, 1: first read of pair is +, 2: first read of pair is -). See ?strandMode. Note: Sets default to 0 instead of 1, as readGAlignmentPairs uses 1. This is to guarantee hits, but will also make mismatches of overlapping transcripts in opposite directions. |
| `seqinfo` | Seqinfo object, defaul NULL (created from ranges). Add to avoid warnings later on differences in seqinfo. |

## Details

Other columns can be named whatever you want and added to meta columns. Positions are 1-based, not 0-based as .bed. Import with import.ofst

## Value

a GAlignment, GAlignmentPairs or GRanges object, dependent of if cigar/cigar1 is defined in .ofst file.

### Examples

```
## GRanges
gr <- GRanges("1:1-3:-")
tmp <- file.path(tempdir(), "path.ofst")
# export.ofst(gr, file = tmp)
# import.ofst(tmp)
## GAlignment
# Make input data.frame
df <- data.frame(seqnames = "1", cigar = "3M", start = 1L, strand = "+")
ga <- ORFik:::getGAlignments(df)
# export.ofst(ga, file = tmp)
# import.ofst(tmp)
```

---

## importGtfFromTxdb *Import the GTF / GFF that made the txdb*

---

### Description

Import the GTF / GFF that made the txdb

### Usage

```
importGtfFromTxdb(txdb, stop.error = TRUE)
```

### Arguments

| | |
|---|---|
| txdb | a TxDb, path to txdb / gff or ORFik experiment object |
| stop.error | logical TRUE, stop if Txdb does not have a gtf. If FALSE, return NULL. |

### Value

data.frame, the gtf/gff object imported with rtracklayer::import. Or NULL, if stop.error is FALSE, and no GTF file found.

---

## initiationScore *Get initiation score for a GRangesList of ORFs*

---

### Description

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

### Usage

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

## Arguments

| | |
|---|---|
| grl | a GRangesList object with ORFs |
| cds | a GRangesList object with coding sequences |
| tx | a GRangesList of transcripts covering grl. |
| reads | ribo seq reads as GAlignments, GRanges or GRangesList object |
| pShifted | a logical (TRUE), are riboseq reads p-shifted? |
| weight | a vector (default: 1L, if 1L it is identical to countOverlaps()), if single number (!= 1), it applies for all, if more than one must be equal size of 'reads'. else it must be the string name of a defined meta column in subject "reads", that gives number of times a read was found. GRanges("chr1", 1, "+", score = 5), would mean "score" column tells that this alignment region was found 5 times. |

## 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

If a score column is defined, it will use it as weights, see getWeights

## Value

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

## References

doi: 10.1186/s12915-017-0416-0

## See Also

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

## Examples

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

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

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

---

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

---

## Description

Inside/Outside score is defined as

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

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

## Usage

```
insideOutsideORF(
  grl,
  RFP,
  GtfOrTx,
  ds = NULL,
  RFP.sorted = FALSE,
  weight = 1L,
  overlapGrl = NULL
)
```

## Arguments

| | |
|---|---|
| `grl` | a `GRangesList` object with usually either leaders, cds', 3' utrs or ORFs. |
| `RFP` | RiboSeq reads as GAlignments, 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` |
| `ds` | numeric vector (NULL), disengagement score. If you have already calculated `disengagementScore`, input here to save time. |
| `RFP.sorted` | logical (FALSE), an optimizer, have you ran this line: `RFP <- sort(RFP[countOverlaps(RFP, tx, type = "within") > 0])` Normally not touched, for internal optimization purposes. |

| | |
|---|---|
| weight | a vector (default: 1L, if 1L it is identical to countOverlaps()), if single number (!= 1), it applies for all, if more than one must be equal size of 'reads'. else it must be the string name of a defined meta column in subject "reads", that gives number of times a read was found. GRanges("chr1", 1, "+", score = 5), would mean "score" column tells that this alignment region was found 5 times. |
| overlapGrl | an integer, (default: NULL), if defined must be countOverlaps(grl, RFP), added for speed if you already have it |

## Value

a named vector of numeric values of scores

## References

doi: 10.1242/dev.098345

## See Also

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

`install.fastp` *Download and prepare fastp trimmer*

## Description

On Linux, will not run "make", only use precompiled fastp file.
On Mac OS it will use precompiled binaries.
For windows must be installed through WSL (Windows Subsystem Linux)

## Usage

```
install.fastp(folder = "~/bin")
```

## Arguments

| | |
|---|---|
| `folder` | path to folder for download, file will be named "fastp", this should be most recent version. On mac it will search for a folder called fastp-master inside folder given. Since there is no precompiled version of fastp for Mac OS. |

## Value

path to runnable fastp

## References

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6129281/

## See Also

Other STAR: `STAR.align.folder()`, `STAR.align.single()`, `STAR.allsteps.multiQC()`, `STAR.index()`, `STAR.install()`, `STAR.multiQC()`, `STAR.remove.crashed.genome()`, `getGenomeAndAnnotation()`

## Examples

```
## With default folder:
#install.fastp()

## Or set manual folder:
folder <- "~/I/WANT/IT/HERE/"
#install.fastp(folder)
```

`install.sratoolkit` *Download sra toolkit*

## Description

Currently supported for Linux (64 bit centos and ubunutu is tested to work) and Mac-OS(64 bit). If other linux distro, centos binaries will be used.

## Usage

```
install.sratoolkit(folder = "~/bin", version = "2.11.3")
```

## Arguments

| | |
|---|---|
| `folder` | default folder, "~/bin" |
| `version` | a string, default "2.11.3" |

## Value

path to fastq-dump in sratoolkit

## References

https://ncbi.github.io/sra-tools/fastq-dump.html

## See Also

Other sra: `browseSRA()`, `download.SRA()`, `download.SRA.metadata()`, `download.ebi()`, `get_bioproject_candidates`
`rename.SRA.files()`

## Examples

```
# install.sratoolkit()
## Custom folder and version (not adviced)
folder <- "/I/WANT/IT/HERE/"
# install.sratoolkit(folder, version = "2.10.9")
```

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

## Description

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

## Usage

```
isInFrame(dists)
```

## Arguments

| | |
|---|---|
| `dists` | a vector of integer distances between ORF and cds. 0 distance means equal frame |

## 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: `computeFeatures()`, `computeFeaturesCage()`, `countOverlapsW()`, `disengagementScore()`, `distToCds()`, `distToTSS()`, `entropy()`, `floss()`, `fpkm()`, `fpkm_calc()`, `fractionLength()`, `initiationScore()`, `insideOutsideORF()`, `isOverlapping()`, `kozakSequenceScore()`, `orfScore()`, `rankOrder()`, `ribosomeReleaseScore()`, `ribosomeStallingScore()`, `startRegion()`, `startRegionCoverage()`, `stopRegion()`, `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: `computeFeatures()`, `computeFeaturesCage()`, `countOverlapsW()`, `disengagementScore()`, `distToCds()`, `distToTSS()`, `entropy()`, `floss()`, `fpkm()`, `fpkm_calc()`, `fractionLength()`, `initiationScore()`, `insideOutsideORF()`, `isInFrame()`, `kozakSequenceScore()`, `orfScore()`, `rankOrder()`, `ribosomeReleaseScore()`, `ribosomeStallingScore()`, `startRegion()`, `startRegionCoverage()`, `stopRegion()`, `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)
```

kozakHeatmap *Make sequence region heatmap relative to scoring*

## Description

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

## Usage

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

## Arguments

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

## Details

It will create blocks around the highest rate per position

## Value

a ggplot of the heatmap

## Examples

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

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

## End(Not run)
```

---

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

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| grl | a GRangesList grouped by ORF |
| tx | a GRangesList, the reference area for ORFs, each ORF must have a corresponding tx. |
| faFile | FaFile, BSgenome, fasta/index file path or an ORFik experiment. This file is usually used to find the transcript sequences from some GRangesList. |

species ("human"), which species to use, currently supports human (Homo sapiens), zebrafish (Danio rerio) and mouse (Mus musculus). Both scientific or common name for these species will work. You can also specify a pfm for your own species. Syntax of pfm is an rectangular integer matrix, where all columns must sum to the same value, normally 100. See example for more information. Rows are in order: c("A", "C", "G", "T")

include.N logical (F), if TRUE, allow N bases to be counted as hits, score will be average of the other bases. If True, N bases will be added to pfm, automaticly, so dont include them if you make your own pfm.

## Details

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

## Value

a numeric vector with values between 0 and 1

an integer vector, one score per orf

## References

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

## See Also

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

## Examples

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

```
kozakSequenceScore(ORFs, tx, faFile)
# For more details see vignettes.
```

---

`kozak_IR_ranking` *Rank kozak initiation sequences*

---

## Description

Defined as region (-4, -1) relative to TIS

## Usage

```
kozak_IR_ranking(cds_k, mrna, dt.ir, faFile, group.min = 10, species = "human")
```

## Arguments

| | |
|---|---|
| `cds_k` | cds ranges (GRangesList) |
| `mrna` | mrna ranges (GRangesList) |
| `dt.ir` | data.table with a column called IR, initiation rate |
| `faFile` | `FaFile`, BSgenome, fasta/index file path or an ORFik `experiment`. This file is usually used to find the transcript sequences from some GRangesList. |
| `group.min` | numeric, default 10. Minimum transcripts per initation group to be included |
| `species` | ("human"), which species to use, currently supports human (Homo sapiens), zebrafish (Danio rerio) and mouse (Mus musculus). Both scientific or common name for these species will work. 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") |

## Value

a ggplot grid object

---

`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, default: (TRUE) |

### 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, default: (TRUE) |

### 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)
```

## length,covRle-method *length covRle*

### Description

Number of chromosomes

### Usage

```
## S4 method for signature 'covRle'
length(x)
```

### Arguments

| | |
|---|---|
| x | a covRle object |

### Value

an integer, number of chromosomes in covRle object

---

`length,covRleList-method` *length covRleList*

---

### Description

Number of covRle objects

### Usage

```
## S4 method for signature 'covRleList'
length(x)
```

### Arguments

| | |
|---|---|
| x | a covRleList object |

### Value

an integer, number of covRle objects

---

`lengths,covRle-method` *lengths covRle*

---

### Description

Lengths of each chromosome

### Usage

```
## S4 method for signature 'covRle'
lengths(x)
```

### Arguments

| | |
|---|---|
| x | a covRle object |

### Value

a named integer vector of chromosome lengths

## lengths,covRleList-method *lengths covRleList*

### Description

Lengths of each chromosome

### Usage

```
## S4 method for signature 'covRleList'
lengths(x)
```

### Arguments

| | |
|---|---|
| x | a covRle object |

### Value

a named integer vector of chromosome lengths

## libFolder *Get ORFik experiment library folder*

### Description

Get ORFik experiment library folder

### Usage

```
libFolder(x, mode = "first")
```

### Arguments

| | |
|---|---|
| x | an ORFik experiment |
| mode | character, default "first". Alternatives: "unique", "all". |

### Value

a character path

---

`libFolder,experiment-method` *Get ORFik experiment library folder*

---

## Description

Get ORFik experiment library folder

## Usage

```
## S4 method for signature 'experiment'
libFolder(x, mode = "first")
```

## Arguments

| | |
|---|---|
| x | an ORFik experiment |
| mode | character, default "first". Alternatives: "unique", "all". |

## Value

a character path

---

`libraryTypes` *Which type of library type in* `experiment`*?*

---

## Description

Which type of library type in experiment?

## Usage

```
libraryTypes(df, uniqueTypes = TRUE)
```

## Arguments

| | |
|---|---|
| df | an ORFik experiment |
| uniqueTypes | logical, default TRUE. Only return unique lib types. |

## Value

library types (character vector)

## See Also

Other ORFik_experiment: ORFik.template.experiment(), ORFik.template.experiment.zf(), bamVarName(), create.experiment(), experiment-class, filepath(), organism,experiment-method, outputLibs(), read.experiment(), save.experiment(), validateExperiments()

## Examples

```
df <- ORFik.template.experiment()
libraryTypes(df)
libraryTypes(df, uniqueTypes = FALSE)
```

---

`list.experiments` *List current experiment available*

---

## Description

Will only search .csv extension, also exclude any experiment with the word template.

## Usage

```
list.experiments(
  dir = ORFik::config()["exp"],
  pattern = "*",
  libtypeExclusive = NULL,
  validate = TRUE,
  BPPARAM = bpparam()
)
```

## Arguments

| | |
|---|---|
| `dir` | directory for ORFik experiments: default: ORFik::config()["exp"], which by default is: "~/Bio_data/ORFik_experiments/" |
| `pattern` | allowed patterns in experiment file name: default ("*", all experiments) |
| `libtypeExclusive` | search for experiments with exclusivly this libtype, default (NULL, all) |
| `validate` | logical, default TRUE. Abort if any library files does not exist. Do not set this to FALSE, unless you know what you are doing! |
| `BPPARAM` | how many cores/threads to use? default: bpparam() |

## Value

a data.table, 1 row per experiment with columns:
- experiment (name),
- organism
- author
- libtypes
- number of samples

## Examples

```
## Make your experiments
df <- ORFik.template.experiment(TRUE)
df2 <- df[1:6,] # Only first 2 libs
## Save them
# save.experiment(df, "~/Bio_data/ORFik_experiments/exp1.csv")
# save.experiment(df2, "~/Bio_data/ORFik_experiments/exp1_subset.csv")
## List all experiment you have:
## Path above is default path, so no dir argument needed
#list.experiments()
#list.experiments(pattern = "subset")
## For non default directory experiments
#list.experiments(dir = "MY/CUSTOM/PATH)
```

---

# list.genomes *List genomes created with ORFik*

---

## Description

Given the reference.folder, list all valid references. An ORFik genome is defined as a folder with a file called output.rds that is a named R vector with names gtf and genome, where the values are character paths to those files inside that folder. This makes sure that this reference was made by ORFik and not some other program.

## Usage

```
list.genomes(reference.folder = ORFik::config()["ref"])
```

## Arguments

`reference.folder`
: character path, default: `ORFik::config()["ref"]`.

## Value

a data.table with 5 columns:
- character (name of folder)
- logical (does it have a gtf)
- logical (does it have a fasta genome)
- logical (does it have a STAR index)
- logical (only displayed if some are TRUE, does it have protein structure predictions of ORFs from alphafold etc, in folder called 'protein_structure_predictions')
- logical (only displayed if some are TRUE, does it have gene symbol fst file from bioMart etc, in file called 'gene_symbol_tx_table.fst')

## Examples

```
## Run with default config path
#list.genomes()
## Run with custom config path
list.genomes(tempdir())
## Get the path to fasta genome of first organism in list
#readRDS(file.path(config()["ref"], list.genomes()$name, "outputs.rds")[1])["genome"]
```

---

`loadRegion` *Load transcript region*

---

## Description

Usefull to simplify loading of standard regions, like cds' and leaders. Adds another safety in that seqlevels will be set

## Usage

```
loadRegion(
  txdb,
  part = "tx",
  names.keep = NULL,
  by = "tx",
  skip.optimized = FALSE
)
```

## Arguments

| | |
|---|---|
| `txdb` | a TxDb file or a path to one of: (.gtf ,.gff, .gff2, .gff2, .db or .sqlite), if it is a GRangesList, it will return it self. |
| `part` | a character, one of: tx, ncRNA, mrna, leader, cds, trailer, intron, NOTE: difference between tx and mrna is that tx are all transcripts, while mrna are all transcripts with a cds, respectivly ncRNA are all tx without a cds. |
| `names.keep` | a character vector of subset of names to keep. Example: loadRegions(txdb, names = "ENST1000005"), will return only that transcript. Remember if you set by to "gene", then this list must be with gene names. |
| `by` | a character, default "tx" Either "tx" or "gene". What names to output region by, the transcript name "tx" or gene names "gene". NOTE: this is not the same as cdsBy(txdb, by = "gene"), cdsBy would then only give 1 cds per Gene, loadRegion gives all isoforms, but with gene names. |
| `skip.optimized` | logical, default FALSE. If TRUE, will not search for optimized rds files to load created from ORFik::makeTxdbFromGenome(..., optimize = TRUE). The optimized files are ~ 100x faster to load for human genome. |

## Details

Load as GRangesList if input is not already GRangesList.

## Value

a GRangesList of region

## Examples

```
# GTF file is slow, but possible to use
gtf <- system.file("extdata", "hg19_knownGene_sample.sqlite",
                        package = "GenomicFeatures")
txdb <- loadTxdb(gtf)
loadRegion(txdb, "cds")
loadRegion(txdb, "intron")
# Use txdb from experiment
df <- ORFik.template.experiment()
txdb <- loadTxdb(df)
loadRegion(txdb, "leaders")
# Use ORFik experiment directly
loadRegion(df, "mrna")
```

---

# loadRegions *Get all regions of transcripts specified to environment*

---

## Description

By default loads all parts to .GlobalEnv (global environemnt) Useful to not spend time on finding the functions to load regions.

## Usage

```
loadRegions(
  txdb,
  parts = c("mrna", "leaders", "cds", "trailers"),
  extension = "",
  names.keep = NULL,
  by = "tx",
  skip.optimized = FALSE,
  envir = .GlobalEnv
)
```

## Arguments

| | |
|---|---|
| txdb | a TxDb file, a path to one of: (.gtf ,.gff, .gff2, .gff2, .db or .sqlite) or an ORFik experiment |
| parts | the transcript parts you want, default: c("mrna", "leaders", "cds", "trailers"). See ?loadRegion for more info on this argument. |
| extension | What to add on the name after leader, like: B -> leadersB |
| names.keep | a character vector of subset of names to keep. Example: loadRegions(txdb, names = "ENST1000005"), will return only that transcript. Remember if you set by to "gene", then this list must be with gene names. |

| | |
|---|---|
| by | a character, default "tx" Either "tx" or "gene". What names to output region by, the transcript name "tx" or gene names "gene". NOTE: this is not the same as cdsBy(txdb, by = "gene"), cdsBy would then only give 1 cds per Gene, loadRegion gives all isoforms, but with gene names. |
| skip.optimized | logical, default FALSE. If TRUE, will not search for optimized rds files to load created from ORFik::makeTxdbFromGenome(..., optimize = TRUE). The optimized files are ~ 100x faster to load for human genome. |
| envir | Which environment to save to, default: .GlobalEnv |

## Value

invisible(NULL) (regions saved in envir)

## Examples

```
# Load all mrna regions to Global environment
gtf <- system.file("extdata", "hg19_knownGene_sample.sqlite",
                        package = "GenomicFeatures")
loadRegions(gtf, parts = c("mrna", "leaders", "cds", "trailers"))
```

---

# loadTranscriptType *Load transcripts of given biotype*

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| object | a TxDb, ORFik experiment or path to gtf/gff, |
| part | a character, default rRNA. Can also be: snoRNA, tRNA etc. As long as that biotype is defined in the gtf. |
| tx | a GRangesList of transcripts (Optional, default NULL, all transcript of that type), else it must be names a list to subset on. |

## Value

a GRangesList of transcript of that type

## References

doi: 10.1002/0471250953.bi0410s25

## Examples

```
gtf <- "path/to.gtf"
#loadTranscriptType(gtf, part = "rRNA")
#loadTranscriptType(gtf, part = "miRNA")
```

---

`loadTxdb` *General loader for txdb*

---

## Description

Useful to allow fast TxDb loader like .db

## Usage

```
loadTxdb(txdb, chrStyle = NULL, organism = NA, chrominfo = NULL)
```

## Arguments

| | |
|---|---|
| `txdb` | a TxDb file, a path to one of: (.gtf ,.gff, .gff2, .gff2, .db or .sqlite) or an ORFik experiment |
| `chrStyle` | a GRanges object, TxDb, FaFile, , a `seqlevelsStyle` or `Seqinfo`. (Default: NULL) to get seqlevelsStyle from. In addition if it is a Seqinfo object, seqinfo will be updated. Example of seqlevelsStyle update: Is chromosome 1 called chr1 or 1, is mitocondrial chromosome called MT or chrM etc. Will use 1st seqlevel-style if more are present. Like: c("NCBI", "UCSC") -> pick "NCBI" |
| `organism` | character, default NA. Scientific name of organism. Only used if input is path to gff. |
| `chrominfo` | Seqinfo object, default NULL. Only used if input is path to gff. |

## Value

a TxDb object

## Examples

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

## longestORFs *Get longest ORF per stop site*

### Description

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

### Usage

```
longestORFs(grl)
```

### Arguments

| | |
|---|---|
| grl | a GRangesList/IRangesList, GRanges/IRanges of ORFs |

### Value

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

### See Also

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

### Examples

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

## 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)
```

---

## makeSummarizedExperimentFromBam

*Make a count matrix from a library or experiment*

---

### Description

Make a summerizedExperiment / matrix object from bam files or other library formats sepcified by lib.type argument. Works like HTSeq, to give you count tables per library.

### Usage

```
makeSummarizedExperimentFromBam(
  df,
  saveName = NULL,
  longestPerGene = FALSE,
  geneOrTxNames = "tx",
  region = "mrna",
  type = "count",
  lib.type = "ofst",
  weight = "score",
  forceRemake = FALSE,
  force = TRUE,
  library.names = bamVarName(df),
  BPPARAM = BiocParallel::SerialParam()
)
```

## Arguments

| | |
|---|---|
| df | an ORFik experiment |
| saveName | a character (default NULL), if set save experiment to path given. Always saved as .rds., it is optional to add .rds, it will be added for you if not present. Also used to load existing file with that name. |
| longestPerGene | a logical (default FALSE), if FALSE all transcript isoforms per gene. Ignored if "region" is not a character of either: "mRNA","tx", "cds", "leaders" or "trailers". |
| geneOrTxNames | a character vector (default "tx"), should row names keep trancript names ("tx") or change to gene names ("gene") |
| region | a character vector (default: "mrna"), make raw count matrices of whole mrnas or one of (leaders, cds, trailers). Can also be a GRangesList, then it uses this region directly. Can then be uORFs or a subset of CDS etc. |
| type | default: "count" (raw counts matrix), alternative is "fpkm", "log2fpkm" or "log10fpkm" |
| lib.type | a character(default: "default"), load files in experiment or some precomputed variant, either "ofst", "bedo", "bedoc" or "pshifted". These are made with ORFik:::convertLibs() or shiftFootprintsByExperiment(). Can also be custom user made folders inside the experiments bam folder. |
| weight | numeric or character, a column to score overlaps by. Default "score", will check for a metacolumn called "score" in libraries. If not found, will not use weights. |
| forceRemake | logical, default FALSE. If TRUE, will not look for existing file count table files. |
| library.names | character, default: bamVarName(df). Names to load libraries as to environment and names to display in plots. |
| BPPARAM | how many cores/threads to use? default: BiocParallel::SerialParam() |

## Details

If txdb or gtf path is added, it is a rangedSummerizedExperiment NOTE: If the file called saveName exists, it will then load file, not remake it!
There are different ways of counting hits on transcripts, ORFik does it as pure coverage (if a single read aligns to a region with 2 genes, both gets a count of 1 from that read). This is the safest way to avoid false negatives (genes with no assigned hits that actually have true hits).

## Value

a SummarizedExperiment object or data.table if "type" is not "count, with rownames as transcript / gene names.

## Examples

```
##Make experiment
df <- ORFik.template.experiment()
# makeSummarizedExperimentFromBam(df)
## Only cds (coding sequences):
# makeSummarizedExperimentFromBam(df, region = "cds")
## FPKM instead of raw counts on whole mrna regions
# makeSummarizedExperimentFromBam(df, type = "fpkm")
```

```
## Make count tables of pshifted libraries over uORFs
uorfs <- GRangesList(uorf1 = GRanges("chr23", 17599129:17599156, "-"))
#saveName <- file.path(dirname(df$filepath[1]), "uORFs", "countTable_uORFs")
#makeSummarizedExperimentFromBam(df, saveName, region = uorfs)
## To load the uORFs later
# countTable(df, region = "uORFs", count.folder = "uORFs")
```

---

`makeTxdbFromGenome` *Make txdb from genome*

---

## Description

Make a Txdb with defined seqlevels and seqlevelsstyle from the fasta genome. This makes it more fail safe than standard Txdb creation. Example is that you can not create a coverage window outside the chromosome boundary, this is only possible if you have set the seqlengths.

## Usage

```
makeTxdbFromGenome(
  gtf,
  genome = NULL,
  organism,
  optimize = FALSE,
  gene_symbols = FALSE,
  uniprot_id = FALSE,
  pseudo_5UTRS_if_needed = NULL,
  return = FALSE
)
```

## Arguments

| | |
|---|---|
| `gtf` | path to gtf file |
| `genome` | character, default NULL. Path to fasta genome corresponding to the gtf. If NULL, can not set seqlevels. If value is NULL or FALSE, it will be ignored. |
| `organism` | Scientific name of organism, first letter must be capital! Example: Homo sapiens. Will force first letter to capital and convert any "_" (underscore) to " " (space) |
| `optimize` | logical, default FALSE. Create a folder within the folder of the gtf, that includes optimized objects to speed up loading of annotation regions from up to 15 seconds on human genome down to 0.1 second. ORFik will then load these optimized objects instead. Currently optimizes filterTranscript() function and loadRegion() function for 5' UTRs, 3' UTRs, CDS, mRNA (all transcript with CDS) and tx (all transcripts). |
| `gene_symbols` | logical default FALSE. If TRUE, will download and store all gene symbols for all transcripts (coding and noncoding)- In a file called: "gene_symbol_tx_table.fst" in same folder as txdb. hgcn for human, mouse symbols for mouse and rat, more to be added. |

| | |
|---|---|
| uniprot_id | logical default FALSE. If TRUE, will download and store all uniprot id for all transcripts (coding and noncoding)- In a file called: "gene_symbol_tx_table.fst" in same folder as txdb. |
| pseudo_5UTRS_if_needed | integer, default NULL. If defined > 0, will add pseudo 5' UTRs if 30 a leader. |
| return | logical, default FALSE. If TRUE, return TXDB object, else NULL. |

## Value

NULL, Txdb saved to disc named paste0(gtf, ".db"). Set 'return' argument to TRUE, to get txdb back

## Examples

```
gtf <- "/path/to/local/annotation.gtf"
genome <- "/path/to/local/genome.fasta"
#makeTxdbFromGenome(gtf, genome, organism = "Saccharomyces cerevisiae")
## Add pseudo UTRs if needed (< 30% of cds have a defined 5'UTR)
```

---

# mergeFastq *Merge groups of Fastq /Fasta files*

---

## Description

Will use multithreading to speed up process. Only works for Unix OS (Linux and Mac)

## Usage

```
mergeFastq(in_files, out_files, BPPARAM = bpparam())
```

## Arguments

| | |
|---|---|
| in_files | character specify the full path to the individual fastq.gz files. Seperated by space per file in group: For 2 output files from 4 input files: in_files <- c("file1.fastq file2.fastq". "file3.fastq file4.fastq") |
| out_files | character specify the path to the FASTQ directory For 2 output files: out_files <- c("/merged/file1&2.fastq", "/merged/file3&4.fastq") |
| BPPARAM | how many cores/threads to use? default: bpparam(). To see number of threads used, do bpparam()$workers |

## Value

invisible(NULL).

## Examples

```
fastq.folder <- tempdir() # <- Your fastq files
infiles <- dir(fastq.folder, "*.fastq", full.names = TRUE)
## Not run:
# Seperate files into groups (here it is 4 output files from 12 input files)
in_files <- c(paste0(grep(infiles, pattern = paste0("ribopool-",
               seq(11, 14), collapse = "|"), value = TRUE), collapse = " "),
              paste0(grep(infiles, pattern = paste0("ribopool-",
               seq(18, 19), collapse = "|"), value = TRUE), collapse = " "),
              paste0(grep(infiles, pattern = paste0("C11-",
               seq(11, 14), collapse = "|"), value = TRUE), collapse = " "),
              paste0(grep(infiles, pattern = paste0("C11-",
               seq(18, 19), collapse = "|"), value = TRUE), collapse = " "))

out_files <- paste0(c("SSU_ribopool", "LSU_ribopool", "SSU_WT", "LSU_WT"), ".fastq.gz")
merged.fastq.folder <- file.path(fastq.folder, "merged/")
out_files <- file.path(merged.fastq.folder, out_files)

mergeFastq(in_files, out_files)

## End(Not run)
```

---

# mergeLibs *Merge and save libraries of experiment*

---

## Description

Aggregate count of reads (from the "score" column) by making a merged library. Only allowed for .ofst files!

## Usage

```
mergeLibs(
  df,
  out_dir = file.path(libFolder(df), "ofst_merged"),
  mode = "all",
  type = "ofst",
  keep_all_scores = TRUE
)
```

## Arguments

| | |
|---|---|
| df | an ORFik experiment |
| out_dir | Ouput directory, default file.path(dirname(df$filepath[1]), "ofst_merged"), saved as "all.ofst" in this folder if mode is "all". Use a folder called pshifted_merged, for default Ribo-seq ofst files. |
| mode | character, default "all". Merge all or "rep" for collapsing replicates only, or "lib" for collapsing all per library type. |

type — a character(default: "default"), load files in experiment or some precomputed variant, like "ofst" or "pshifted". These are made with ORFik:::convertLibs(), shiftFootprintsByExperiment(), etc. Can also be custom user made folders inside the experiments bam folder. It acts in a recursive manner with priority: If you state "pshifted", but it does not exist, it checks "ofst". If no .ofst files, it uses "default", which always must exists.
Presets are (folder is relative to default lib folder, some types fall back to other formats if folder does not exist):
- "default": load the original files for experiment, usually bam.
- "ofst": loads ofst files from the ofst folder, relative to lib folder (falls back to default)
- "pshifted": loads ofst, wig or bigwig from pshifted folder (falls back to ofst, then default)
- "cov": Load covRle objects from cov_RLE folder (fail if not found)
- "covl": Load covRleList objects, from cov_RLE_List folder (fail if not found)
- "bed": Load bed files, from bed folder (falls back to default)
- Other formats must be loaded directly with fimport

keep_all_scores — logical, default TRUE, keep all library scores in the merged file. These score columns are named the libraries full name from `bamVarName(df)`.

## Value

NULL, files saved to disc. A data.table with a score column that now contains the sum of scores per merge setting.

## Examples

```
df2 <- ORFik.template.experiment()
df2 <- df2[df2$libtype == "RFP",]
# Merge all
#mergeLibs(df2, tempdir(), mode = "all", type = "default")
# Read as GRanges with mcols
#fimport(file.path(tempdir(), "all.ofst"))
# Read as direct fst data.table
#read_fst(file.path(tempdir(), "all.ofst"))
# Collapse replicates
#mergeLibs(df2, tempdir(), mode = "rep", type = "default")
# Collapse by lib types
#mergeLibs(df2, tempdir(), mode = "lib", type = "default")
```

---

# metadata.autnaming *Guess SRA metadata columns*

---

## Description

Guess SRA metadata columns

### Usage

```
metadata.autnaming(file)
```

### Arguments

| | |
|---|---|
| file | a data.table of SRA metadata |

### Value

a data.table of SRA metadata with additional columns: LIBRARYTYPE, REPLICATE, STAGE, CONDITION, INHIBITOR

---

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

---

### Description

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

### Usage

```
metaWindow(
  x,
  windows,
  scoring = "sum",
  withFrames = FALSE,
  zeroPosition = NULL,
  scaleTo = 100,
  fraction = NULL,
  feature = NULL,
  forceUniqueEven = !is.null(scoring),
  forceRescale = TRUE,
  weight = "score",
  drop.zero.dt = FALSE,
  append.zeroes = FALSE
)
```

### Arguments

| | |
|---|---|
| x | GRanges/GAlignment object of your reads. Remember to resize them beforehand to width of 1 to focus on 5' ends of footprints etc, if that is wanted. |
| windows | GRangesList or GRanges of your ranges |
| scoring | a character, default: "sum", one of (zscore, transcriptNormalized, mean, median, sum, sumLength, NULL), see ?coverageScorings for info and more alternatives. |
| withFrames | a logical (TRUE), return positions with the 3 frames, relative to zeroPosition. zeroPosition is frame 0. |

`zeroPosition` an integer DEFAULT (NULL), the point if all windows are equal size, that should be set to position 0. Like leaders and cds combination, then 0 is the TIS and -1 is last base in leader. NOTE!: if not all windows have equal width, this will be ignored. If all have equal width and zeroPosition is NULL, it is set to as.integer(width / 2).

`scaleTo` an integer (100), if windows have different size, a meta window can not directly be created, since a meta window must have equal size for all windows. Rescale (bin) all windows to scaleTo. i.e c(1,2,3) -> size 2 -> coverage of position c(1, mean(2,3)) etc.

`fraction` a character/integer (NULL), the fraction i.e (27) for read length 27, or ("LSU") for large sub-unit TCP-seq.

`feature` a character string, info on region. Usually either gene name, transcript part like cds, leader, or CpG motifs etc.

`forceUniqueEven` a logical (TRUE), if TRUE; require that all windows are of same width and even. To avoid bugs. FALSE if score is NULL.

`forceRescale` logical, default TRUE. If TRUE, if `unique(widthPerGroup(windows))` has length > 1, it will force all windows to width of the `scaleTo` argument, making a binned meta coverage.

`weight` (default: 'score'), if defined a character name of valid meta column in subject. GRanges("chr1", 1, "+", score = 5), would mean score column tells that this alignment region was found 5 times. ORFik ofst, bedoc and .bedo files contains a score column like this. As do CAGEr CAGE files and many other package formats. You can also assign a score column manually.

`drop.zero.dt` logical FALSE, if TRUE and as.data.table is TRUE, remove all 0 count positions. This greatly speeds up and most importantly, greatly reduces memory usage. Will not change any plots, unless 0 positions are used in some sense. (mean, median, zscore coverage will only scale differently)

`append.zeroes` logical, default FALSE. If TRUE and drop.zero.dt is TRUE and all windows have equal length, it will add back 0 values after transformation. Sometimes needed for correct plots, if TRUE, will call abort if not all windows are equal length!

## Value

A data.table with scored counts (score) of reads mapped to positions (position) specified in windows along with frame (frame) per gene (genes) per library (fraction) per transcript region (feature). Column that does not apply is not given, but position and (score/count) is always returned.

## See Also

Other coverage: `coverageScorings()`, `regionPerReadLength()`, `scaledWindowPositions()`, `windowPerReadLength()`

## Examples

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

---

`model.matrix,experiment-method`

*Get experiment design model matrix*

---

## Description

The function extends stats::model.matrix.

## Usage

```
## S4 method for signature 'experiment'
model.matrix(object, design_formula = design(object, as.formula = TRUE))
```

## Arguments

| | |
|---|---|
| `object` | an ORFik `experiment` |
| `design_formula` | the experiment design, as formula, subset columns, to change the model.matrix, default: `design(object, as.formula = TRUE)` |

## Value

a matrix with design and level attributes

## Examples

```
df <- ORFik.template.experiment()
model.matrix(df)
```

name *Get name of ORFik experiment*

## Description

Get name of ORFik experiment

## Usage

```
name(x)
```

## Arguments

| | |
|---|---|
| x | an ORFik experiment |

## Value

character, name of experiment

name,experiment-method *Get name of ORFik experiment*

## Description

Get name of ORFik experiment

## Usage

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

## Arguments

| | |
|---|---|
| x | an ORFik experiment |

## Value

character, name of experiment

## nrow,experiment-method

*Internal nrow function for ORFik experiment Number of runs in experiment*

### Description

Internal nrow function for ORFik experiment Number of runs in experiment

### Usage

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

### Arguments

| | |
|---|---|
| x | an ORFik experiment |

### Value

number of rows in experiment (integer)

## 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 logical, keep names or not, default: (TRUE) |

### 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)
```

---

`ofst_merge` *Merge multiple ofst file*

---

## Description

Collapses and sums the score column of each ofst file It is required that each file is of same ofst type. That is if one file has cigar information, all must have it.

## Usage

```
ofst_merge(
  file_paths,
  lib_names = sub(pattern = "\\.ofst$", replacement = "", basename(file_paths)),
  keep_all_scores = TRUE,
  keepCigar = TRUE,
  sort = TRUE
)
```

## Arguments

| | |
|---|---|
| `file_paths` | Full path to .ofst files wanted to merge |
| `lib_names` | the name to give the resulting score columns |
| `keep_all_scores` | logical, default TRUE, keep all library scores in the merged file. These score columns are named the libraries full name from `bamVarName(df)`. |
| `keepCigar` | logical, default TRUE. If CIGAR is defined, keep column. Setting to FALSE compresses the file much more usually. |
| `sort` | logical, default TRUE. Sort the ranges. Will make the file smaller and faster to load, but some additional merging time is added. |

## Value

a data.table of merged result, it is merged on all columns except "score". The returned file will contain the scores of each file + the aggregate sum score.

optimizedTranscriptLengths
*Load length and names of all transcripts*

## Description

A speedup wrapper around transcriptLengths, default load time of lengths is ~ 15 seconds, if ORFik fst optimized lengths object has been made, load that file instead: load time reduced to ~ 0.1 second.

## Usage

```
optimizedTranscriptLengths(
  txdb,
  with.utr5_len = TRUE,
  with.utr3_len = TRUE,
  create.fst.version = FALSE
)
```

## Arguments

| | |
|---|---|
| txdb | a TxDb file or a path to one of: (.gtf ,.gff, .gff2, .gff2, .db or .sqlite), if it is a GRangesList, it will return it self. |
| with.utr5_len | logical TRUE, include length of 5' UTRs, ignored if .fst exists |
| with.utr3_len | logical TRUE, include length of 3' UTRs, ignored if .fst exists |
| create.fst.version | logical, FALSE. If TRUE, creates a .fst version of the transcript length table (if it not already exists), reducing load time from ~ 15 seconds to ~ 0.01 second next time you run filterTranscripts with this txdb object. The file is stored in the same folder as the genome this txdb is created from, with the name: `paste0(ORFik:::remove.file_ext(metadata(txdb)[3,2]), "_", gsub(" \(.*\|\|:", "", metadata(txdb)[metadata(txdb)[,1] == "Creation time",2]), "_txLengths.fst")` Some error checks are done to see this is a valid location, if the txdb data source is a repository like UCSC and not a local folder, it will not be made. |

## Value

a data.table of loaded lengths 8 columns, 1 row per transcript isoform.

## Examples

```
dt <- optimizedTranscriptLengths(ORFik.template.experiment())
dt
dt[cds_len > 0,] # All mRNA
```

`orfFrameDistributions` *Find shifted Ribo-seq frame distributions*

## Description

Per library: get coverage over CDS per frame per readlength Return as data.datable with information and best frame found. Can be used to automize re-shifting of read lengths (find read lengths where frame 0 is not the best frame over the entire cds)

## Usage

```
orfFrameDistributions(
  df,
  type = "pshifted",
  weight = "score",
  orfs = loadRegion(df, part = "cds"),
  BPPARAM = BiocParallel::bpparam()
)
```

## Arguments

| | |
|---|---|
| `df` | an ORFik `experiment` |
| `type` | type of library loaded, default pshifted, warning if not pshifted might crash if too many read lengths! |
| `weight` | which column in reads describe duplicates, default "score". |
| `orfs` | GRangesList, default loadRegion(df, part = "cds") |
| `BPPARAM` | how many cores/threads to use? default: bpparam(). To see number of threads used, do `bpparam()$workers`. You can also add a time remaining bar, for a more detailed pipeline. |

## Value

data.table with columns: fraction (library) frame (0, 1, 2) score (coverage) length (read length) percent (coverage percentage of library) percent_length (coverage percentage of library and length) best_frame (TRUE/FALSE, is this the best frame per length)

## Examples

```
df <- ORFik.template.experiment()[3,]
dt <- orfFrameDistributions(df, BPPARAM = BiocParallel::SerialParam())
## Check that frame 0 is best frame for all
all(dt[frame == 0,]$best_frame)
```

`ORFik.template.experiment` *An ORFik experiment to see how it looks*

## Description

Toy-data created to resemble human genes:
Number of genes: 6
Genome size: 1161nt x 6 chromosomes = 6966 nt
Experimental design (2 replicates, Wild type vs Mutant)
CAGE: 4 libraries
PAS (poly-A): 4 libraries
Ribo-seq: 4 libraries
RNA-seq: 4 libraries

## Usage

```
ORFik.template.experiment(as.temp = FALSE)
```

## Arguments

| | |
|---|---|
| `as.temp` | logical, default FALSE, load as ORFik experiment. If TRUE, loads as data.frame template of the experiment. |

## Value

an ORFik `experiment`

## See Also

Other ORFik_experiment: `ORFik.template.experiment.zf()`, `bamVarName()`, `create.experiment()`, `experiment-class`, `filepath()`, `libraryTypes()`, `organism`, `experiment-method`, `outputLibs()`, `read.experiment()`, `save.experiment()`, `validateExperiments()`

## Examples

```
ORFik.template.experiment()
```

## ORFik.template.experiment.zf

*An ORFik experiment to see how it looks*

### Description

Toy-data created to resemble Zebrafish genes:
Number of genes: 150
Ribo-seq: 1 library

### Usage

```
ORFik.template.experiment.zf(as.temp = FALSE)
```

### Arguments

| | |
|---|---|
| as.temp | logical, default FALSE, load as ORFik experiment. If TRUE, loads as data.frame template of the experiment. |

### Value

an ORFik experiment

### See Also

Other ORFik_experiment: ORFik.template.experiment(), bamVarName(), create.experiment(), experiment-class, filepath(), libraryTypes(), organism, experiment-method, outputLibs(), read.experiment(), save.experiment(), validateExperiments()

### Examples

```
ORFik.template.experiment.zf()
```

## ORFikQC

*A post Alignment quality control of reads*

### Description

The ORFik QC uses the aligned files (usually bam files), fastp and STAR log files combined with annotation to create relevant statistics.

This report consists of several steps:
1. Convert bam file / Input files to ".ofst" format, if not already done. This format is around 400x faster to use in R than the bam format. Files are also outputted to R environment specified by envExp(df)
2. From this report you will get a summary csv table, with distribution of aligned reads and overlap

counts over transcript regions like: leader, cds, trailer, lincRNAs, tRNAs, rRNAs, snoRNAs etc. It will be called STATS.csv. And can be imported with QCstats function.
3. It will also make correlation plots and meta coverage plots, so you get a good understanding of how good the quality of your NGS data production + aligner step were.
4. Count tables are produced, similar to HTseq count tables. Over mrna, leader, cds and trailer separately. This tables are stored as SummarizedExperiment, for easy loading into DEseq, conversion to normalized fpkm values, or collapsing replicates in an experiment. And can be imported with countTable function.

Everything will be outputed in the directory of your NGS data, inside the folder ./QC_STATS/, relative to data location in 'df'. You can specify new out location with out.dir if you want.
To make a ORFik experiment, see ?ORFik::experiment
To see some normal mrna coverage profiles of different RNA-seq protocols: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4310221/figure/F6/

## Usage

```
ORFikQC(
  df,
  out.dir = resFolder(df),
  plot.ext = ".pdf",
  create.ofst = TRUE,
  complex.correlation.plots = TRUE,
  library.names = bamVarName(df),
  use_simplified_reads = TRUE,
  BPPARAM = bpparam()
)
```

## Arguments

| | |
|---|---|
| df | an ORFik experiment |
| out.dir | character, output directory, default: resFolder(df). Will make a folder within this called "QC_STATS" with all results in this directory. Warning: If you assign not default path, you will have a hazzle to load files later. Much easier to load count tables, statistics, ++ later with default. Update resFolder of df instead if needed. |
| plot.ext | character, default: ".pdf". Alternatives: ".png" or ".jpg". Note that in pdf format the complex correlation plots become very slow to load! |
| create.ofst | logical, default TRUE. Create ".ofst" files from the input libraries, ofst is much faster to load in R, for later use. Stored in ./ofst/ folder relative to experiment main folder. |
| complex.correlation.plots | logical, default TRUE. Add in addition to simple correlation plot two computationally heavy dots + correlation plots. Useful for deeper analysis, but takes longer time to run, especially on low-quality gpu computers. Set to FALSE to skip these. |
| library.names | character, default: bamVarName(df). Names to load libraries as to environment and names to display in plots. |

use_simplified_reads
: logical, default TRUE. For count tables and coverage plots a speed up for GAlignments is to use 5' ends only. This will lose some detail for splice sites, but is usually irrelevant. Note: If reads are precollapsed GRanges, set to FALSE to avoid recollapsing.

BPPARAM
: how many cores/threads to use? default: bpparam(). To see number of threads used, do `bpparam()$workers`. You can also add a time remaining bar, for a more detailed pipeline.

## Value

invisible(NULL) (objects are stored to disc)

## See Also

Other QC report: `QCplots()`, `QCstats()`

## Examples

```
# Load an experiment
df <- ORFik.template.experiment()
# Run QC
#QCreport(df, tempdir())
# QC on subset
#QCreport(df[9,], tempdir())
```

---

# 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. IMPORTANT: Only use p-shifted libraries, see (`detectRibosomeShifts`). Else this score makes no sense.

## Usage

```
orfScore(
  grl,
  RFP,
  is.sorted = FALSE,
  weight = "score",
  overlapGrl = NULL,
  coverage = NULL,
  stop3 = TRUE
)
```

## Arguments

| | |
|---|---|
| grl | a GRangesList of 5' utrs, CDS, transcripts, etc. |
| RFP | ribosomal footprints, given as GAlignments or GRanges object, must be already shifted and resized to the p-site. Requires a $size column with original read lengths. |
| is.sorted | logical (FALSE), is grl sorted. That is + strand groups in increasing ranges (1,2,3), and - strand groups in decreasing ranges (3,2,1) |
| weight | (default: 'score'), if defined a character name of valid meta column in subject. GRanges("chr1", 1, "+", score = 5), would mean score column tells that this alignment region was found 5 times. ORFik ofst, bedoc and .bedo files contains a score column like this. As do CAGEr CAGE files and many other package formats. You can also assign a score column manually. |
| overlapGrl | an integer, (default: NULL), if defined must be countOverlaps(grl, RFP), added for speed if you already have it |
| coverage | a data.table from coveragePerTiling of length same as 'grl' argument. Save time if you have already computed it. |
| stop3 | logical, default TRUE. Stop if any input is of width < 3. |

## Details

Pseudocode: assume rff - is reads fraction in specific frame

```
ORFScore = log(rff1 + rff2 + rff3)
```

If rff2 or rff3 is bigger than rff1, negate the resulting value.

```
ORFScore[rff1Smaller] <- ORFScore[rff1Smaller] * -1
```

As result there is one value per ORF: - Positive values say that the first frame have the most reads, - zero values means it is uniform: (ORFscore between -2.5 and 2.5 can be considered close to uniform), - negative values say that the first frame does not have the most reads. NOTE non-pshifted reads: If reads are not of width 1, then a read from 1-4 on range of 1-4, will get scores frame1 = 2, frame2 = 1, frame3 = 1. What could be logical is that only the 5' end is important, so that only frame1 = 1, to get this, you first resize reads to 5'end only.

General NOTES: 1. p shifting is not exact, so some functional ORFs will get a bad ORF score.
2. If a score column is defined, it will use it as weights, set to weight = 1L if you don't have weight, and score column is something else. 3. If needed a test for significance and critical values, use chi-squared. There are 3 degrees of freedom (3 frames), so critical 0.05 (3-1 degrees of freedm = 2), value is: log2(6) = 2.58 see getWeights

## Value

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

## References

doi: 10.1002/embj.201488411

## See Also

Other features: computeFeatures(), computeFeaturesCage(), countOverlapsW(), disengagementScore(), distToCds(), distToTSS(), entropy(), floss(), fpkm(), fpkm_calc(), fractionLength(), initiationScore(), insideOutsideORF(), isInFrame(), isOverlapping(), kozakSequenceScore(), rankOrder(), ribosomeReleaseScore(), ribosomeStallingScore(), startRegion(), startRegionCoverage(), stopRegion(), 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)
```

---

# organism,experiment-method

*Get ORFik experiment organism*

---

## Description

If not defined directly, checks the txdb / gtf organism information, if existing.

## Usage

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

## Arguments

| | |
|---|---|
| object | an ORFik experiment |

## Value

character, name of organism

## See Also

Other ORFik_experiment: ORFik.template.experiment(), ORFik.template.experiment.zf(), bamVarName(), create.experiment(), experiment-class, filepath(), libraryTypes(), outputLibs(), read.experiment(), save.experiment(), validateExperiments()

## Examples

```
# if you have set organism in txdb of ORFik experiment:
df <- ORFik.template.experiment()
organism(df)

#' If you have not set the organism you can do:
#gtf <- "pat/to/gff_or_gff"
#txdb_path <- paste0(gtf, ".db") # This file is created in next step
#txdb <- makeTxdbFromGenome(gtf, genome, organism = "Homo sapiens",
# optimize = TRUE, return = TRUE)
# then use this txdb in you ORFik experiment and load:
# create.experiment(exper = "new_experiment",
#   txdb = txdb_path) ...
# organism(read.experiment("new-experiment))
```

---

# outputLibs *Output NGS libraries to R as variables*

---

## Description

By default loads the original files of the experiment into the global environment, named by the rows of the experiment required to make all libraries have unique names.
Uses multiple cores to load, defined by multicoreParam

## Usage

```
outputLibs(
  df,
  type = "default",
  paths = filepath(df, type),
  param = NULL,
  strandMode = 0,
  naming = "minimum",
  library.names = name_decider(df, naming),
  output.mode = "envir",
  chrStyle = NULL,
  envir = envExp(df),
  verbose = TRUE,
  force = TRUE,
  BPPARAM = bpparam()
)
```

## Arguments

| | |
|---|---|
| df | an ORFik experiment |

`type` a character(default: "default"), load files in experiment or some precomputed variant, like "ofst" or "pshifted". These are made with ORFik:::convertLibs(), shiftFootprintsByExperiment(), etc. Can also be custom user made folders inside the experiments bam folder. It acts in a recursive manner with priority: If you state "pshifted", but it does not exist, it checks "ofst". If no .ofst files, it uses "default", which always must exists.
Presets are (folder is relative to default lib folder, some types fall back to other formats if folder does not exist):
- "default": load the original files for experiment, usually bam.
- "ofst": loads ofst files from the ofst folder, relative to lib folder (falls back to default)
- "pshifted": loads ofst, wig or bigwig from pshifted folder (falls back to ofst, then default)
- "cov": Load covRle objects from cov_RLE folder (fail if not found)
- "covl": Load covRleList objects, from cov_RLE_List folder (fail if not found)
- "bed": Load bed files, from bed folder (falls back to default)
- Other formats must be loaded directly with fimport

`paths` character vector, the filpaths to use, default `filepath(df, type)`. Change type argument if not correct. If that is not enough, then you can also update this argument. But be careful about using this directly.

`param` `NULL` or a ScanBamParam object. Like for `scanBam`, this influences what fields and which records are imported. However, note that the fields specified thru this ScanBamParam object will be loaded *in addition* to any field required for generating the returned object (GAlignments, GAlignmentPairs, or GappedReads object), but only the fields requested by the user will actually be kept as metadata columns of the object.
By default (i.e. `param=NULL` or `param=ScanBamParam()`), no additional field is loaded. The flag used is `scanBamFlag(isUnmappedQuery=FALSE)` for `readGAlignments`, `readGAlignmentsList`, and `readGappedReads`. (i.e. only records corresponding to mapped reads are loaded), and `scanBamFlag(isUnmappedQuery=FALSE, isPaired=TRUE, hasUnmappedMate=FALSE)` for `readGAlignmentPairs` (i.e. only records corresponding to paired-end reads with both ends mapped are loaded).

`strandMode` numeric, default 0. Only used for paired end bam files. One of (0: strand = *, 1: first read of pair is +, 2: first read of pair is -). See ?strandMode. Note: Sets default to 0 instead of 1, as readGAlignmentPairs uses 1. This is to guarantee hits, but will also make mismatches of overlapping transcripts in opposite directions.

`naming` a character (default: "minimum"). Name files as minimum information needed to make all files unique. Set to "full" to get full names. Set to "fullexp", to get full name with experiment name as prefix, the last one guarantees uniqueness.

`output.mode` character, default "envir". Output libraries to environment. Alternative: "list", return as list. "envirlist", output to envir and return as list. If output is list format, the list elements are named from: `bamVarName(df.rfp)` (Full or minimum naming based on 'naming' argument)

`chrStyle` a GRanges object, TxDb, FaFile, , a `seqlevelsStyle` or `Seqinfo`. (Default: NULL) to get seqlevelsStyle from. In addition if it is a Seqinfo object, seqinfo will be updated. Example of seqlevelsStyle update: Is chromosome 1 called

| | |
|---|---|
| | chr1 or 1, is mitocondrial chromosome called MT or chrM etc. Will use 1st seqlevel-style if more are present. Like: c("NCBI", "UCSC") -> pick "NCBI" |
| envir | environment to save to, default `envExp(df)`, which defaults to .GlobalEnv, but can be set with `envExp(df) <- new.env()` etc. |
| verbose | logical, default TRUE, message about library output status. |
| force | logical, default TRUE If TRUE, reload library files even if matching named variables are found in environment used by experiment (see `envExp`) A simple way to make sure correct libraries are always loaded. FALSE is faster if data is loaded correctly already. |
| BPPARAM | how many cores/threads to use? default: bpparam(). To see number of threads used, do `bpparam()$workers`. You can also add a time remaining bar, for a more detailed pipeline. |

## Details

The functions checks if the total set of libraries have already been loaded: i.e. Check if all names from 'library.names' exists as S4 objects in environment of experiment.

## Value

NULL (libraries set by envir assignment), unless output.mode is "list" or "envirlist": Then you get a list of the libraries.

## See Also

Other ORFik_experiment: `ORFik.template.experiment()`, `ORFik.template.experiment.zf()`, `bamVarName()`, `create.experiment()`, `experiment-class`, `filepath()`, `libraryTypes()`, `organism,experiment-meth`
`read.experiment()`, `save.experiment()`, `validateExperiments()`

## Examples

```
## Load a template ORFik experiment
df <- ORFik.template.experiment()
## Default library type load, usually bam files
# outputLibs(df, type = "default")
## .ofst file load, if ofst files does not exists
## it will load default
# outputLibs(df, type = "ofst")
## .wig file load, if wiggle files does not exists
## it will load default
# outputLibs(df, type = "wig")
## Load as list
outputLibs(df, output.mode = "list")
## Load libs to new environment (called ORFik in Global)
# outputLibs(df, envir = assign(name(df), new.env(parent = .GlobalEnv)))
## Load to hidden environment given by experiment
# envExp(df) <- new.env()
# outputLibs(df)
```

pcaExperiment *Simple PCA analysis*

## Description

Detect outlier libraries with PCA analysis. Will output PCA plot of PCA component 1 (x-axis) vs PCA component 2 (y-axis) for each library (colored by library), shape by replicate. Will be extended to allow batch correction in the future.

## Usage

```
pcaExperiment(
  df,
  output.dir = NULL,
  table = countTable(df, "cds", type = "fpkm"),
  title = "PCA analysis by CDS fpkm",
  subtitle = paste("Numer of genes/regions:", nrow(table)),
  plot.ext = ".pdf",
  return.data = FALSE,
  color.by.group = TRUE
)
```

## Arguments

| | |
|---|---|
| df | an ORFik experiment |
| output.dir | default NULL, else character path to directory. File saved as "PCAplot_(experiment name)(plot.ext)" |
| table | data.table, default countTable(df, "cds", type = "fpkm"), a data.table of counts per column (default normalized fpkm values). |
| title | character, default "CDS fpkm". |
| subtitle | character, default: paste("Numer of genes:", nrow(table)) |
| plot.ext | character, default: ".pdf". Alternatives: ".png" or ".jpg". Note that in pdf format the complex correlation plots become very slow to load! |
| return.data | logical, default FALSE. Return data instead of plot |
| color.by.group | logical, default TRUE. Colors in PCA plot represent unique library groups, if FALSE. Color each sample in seperate color (harder to distinguish for > 10 samples) |

## Value

ggplot or invisible(NULL) if output.dir is defined or < 3 samples. Returns data.table with PCA analysis if return.data is TRUE.

## Examples

```
df <- ORFik.template.experiment()
# Select only Ribo-seq and RNA-seq
pcaExperiment(df[df$libtype %in% c("RNA", "RFP"),])
```

---

pmapFromTranscriptF *Faster pmapFromTranscript*

---

## Description

Map range coordinates between features in the transcriptome and genome (reference) space. The length of x must be the same as length of transcripts. Only exception is if x have integer names like (1, 3, 3, 5), so that x[1] maps to 1, x[2] maps to transcript 3 etc.

## Usage

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

## Arguments

| | |
|---|---|
| x | IRangesList/IRanges/GRanges to map to genomic coordinates |
| transcripts | a GRangesList to map against (the genomic coordinates) |
| removeEmpty | a logical, remove non hit exons, else they are set to 0. That is all exons in the reference that the transcript coordinates do not span. |

## Details

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

## Value

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

## Examples

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

`pmapToTranscriptF` *Faster pmapToTranscript*

## Description

Map range coordinates between features in the transcriptome and genome (reference) space. The length of x must be the same as length of transcripts. Only exception is if x have integer names like (1, 3, 3, 5), so that x[1] maps to 1, x[2] maps to transcript 3 etc.

## Usage

```
pmapToTranscriptF(
  x,
  transcripts,
  ignore.strand = FALSE,
  x.is.sorted = TRUE,
  tx.is.sorted = TRUE
)
```

## Arguments

| | |
|---|---|
| `x` | GRangesList/GRanges/IRangesList/IRanges to map to transcriptomic coordinates |
| `transcripts` | a GRangesList/GRanges/IRangesList/IRanges to map against (the genomic coordinates). Must be of lower abstraction level than x. So if x is GRanges, transcripts can not be IRanges etc. |
| `ignore.strand` | When ignore.strand is TRUE, strand is ignored in overlaps operations (i.e., all strands are considered "+") and the strand in the output is '*'.<br>When ignore.strand is FALSE (default) strand in the output is taken from the transcripts argument. When transcripts is a GRangesList, all inner list elements of a common list element must have the same strand or an error is thrown.<br>Mapped position is computed by counting from the transcription start site (TSS) and is not affected by the value of ignore.strand. |
| `x.is.sorted` | if x is a GRangesList object, are "-" strand groups pre-sorted in decreasing order within group, default: TRUE |
| `tx.is.sorted` | if transcripts is a GRangesList object, are "-" strand groups pre-sorted in decreasing order within group, default: TRUE |

## Details

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

## Value

object of same class as input x, names from ranges are kept.

## Examples

```
library(GenomicFeatures)
# Need 2 ranges object, the target region and whole transcript
# x is target region
x <- GRanges("chr1", IRanges(start = c(26, 29), end = c(27, 29)), "+")
names(x) <- rep("tx1_ORF1", length(x))
x <- groupGRangesBy(x)
# tx is the whole region
tx_gr <- GRanges("chr1", IRanges(c(5, 29), c(27, 30)), "+")
names(tx_gr) <- rep("tx1", length(tx_gr))
tx <- groupGRangesBy(tx_gr)
pmapToTranscriptF(x, tx)
pmapToTranscripts(x, tx)

# Reuse names for matching
x <- GRanges("chr1", IRanges(start = c(26, 29, 5), end = c(27, 29, 18)), "+")
names(x) <- c(rep("tx1_1", 2), "tx1_2")
x <- groupGRangesBy(x)
tx1_2 <- GRanges("chr1", IRanges(c(4, 28), c(26, 31)), "+")
names(tx1_2) <- rep("tx1", 2)
tx <- c(tx, groupGRangesBy(tx1_2))

a <- pmapToTranscriptF(x, tx[txNames(x)])
b <- pmapToTranscripts(x, tx[txNames(x)])
identical(a, b)
seqinfo(a)
# A note here, a & b only have 1 seqlength, even though the 2 "tx1"
# are different in size. This is an artifact of using duplicated names.

## Also look at the asTx for a similar useful function.
```

---

`pSitePlot` *Plot area around TIS as histogram*

---

## Description

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

## Usage

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

```
  title = "auto",
  facet = FALSE,
  frameSum = FALSE
)
```

## Arguments

| | |
|---|---|
| hitMap | a data.frame/data.table, given from metaWindow (must have columns: position, (score or count) and frame) |
| length | an integer (29), which read length is this for? |
| region | a character (start), either "start or "stop" |
| output | character (NULL), if set, saves the plot as pdf or png to path given. If no format is given, is save as pdf. |
| type | character (canonical CDS), type for plot |
| scoring | character, default: (Averaged counts), which scoring did you use ? see ?coverageScorings for info and more alternatives. |
| forHeatmap | a logical (FALSE), should the plot be part of a heatmap? It will scale it differently. Removing title, x and y labels, and truncate spaces between bars. |
| title | character, title of plot. Default "auto", will make it: paste("Length", length, "over", region, "of", type). Else set your own (set to NULL to remove all together). |
| facet | logical, default FALSE. If you input multiple read lengths, specified by fraction column of hitMap, it will split the plots for each read length, putting them under each other. Ignored if forHeatmap is TRUE. |
| frameSum | logical default FALSE. If TRUE, add an addition plot to the right, sum per frame over all positions per length. |

## Details

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

## Value

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

## See Also

Other coveragePlot: `coverageHeatMap()`, `savePlot()`, `windowCoveragePlot()`

## Examples

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

# See vignette for more examples
```

---

# QCfolder *Get ORFik experiment QC folder path*

---

## Description

Get ORFik experiment QC folder path

## Usage

```
QCfolder(x)
```

## Arguments

| | |
|---|---|
| x | an ORFik experiment |

## Value

a character path

---

# QCfolder,experiment-method *Get ORFik experiment QC folder path*

---

## Description

Get ORFik experiment QC folder path

## Usage

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

### Arguments

| | |
|---|---|
| x | an ORFik experiment |

### Value

a character path

---

`QCreport` *A post Alignment quality control of reads*

---

### Description

The ORFik QC uses the aligned files (usually bam files), fastp and STAR log files combined with annotation to create relevant statistics.

This report consists of several steps:
1. Convert bam file / Input files to ".ofst" format, if not already done. This format is around 400x faster to use in R than the bam format. Files are also outputted to R environment specified by `envExp(df)`
2. From this report you will get a summary csv table, with distribution of aligned reads and overlap counts over transcript regions like: leader, cds, trailer, lincRNAs, tRNAs, rRNAs, snoRNAs etc. It will be called STATS.csv. And can be imported with `QCstats` function.
3. It will also make correlation plots and meta coverage plots, so you get a good understanding of how good the quality of your NGS data production + aligner step were.
4. Count tables are produced, similar to HTseq count tables. Over mrna, leader, cds and trailer separately. This tables are stored as `SummarizedExperiment`, for easy loading into DEseq, conversion to normalized fpkm values, or collapsing replicates in an experiment. And can be imported with `countTable` function.

Everything will be outputed in the directory of your NGS data, inside the folder ./QC_STATS/, relative to data location in 'df'. You can specify new out location with out.dir if you want.
To make a ORFik experiment, see ?ORFik::experiment
To see some normal mrna coverage profiles of different RNA-seq protocols: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4310221/figure/F6/

### Usage

```
QCreport(
  df,
  out.dir = resFolder(df),
  plot.ext = ".pdf",
  create.ofst = TRUE,
  complex.correlation.plots = TRUE,
  library.names = bamVarName(df),
  use_simplified_reads = TRUE,
  BPPARAM = bpparam()
)
```

## Arguments

| | |
|---|---|
| df | an ORFik experiment |
| out.dir | character, output directory, default: resFolder(df). Will make a folder within this called "QC_STATS" with all results in this directory. Warning: If you assign not default path, you will have a hazzle to load files later. Much easier to load count tables, statistics, ++ later with default. Update resFolder of df instead if needed. |
| plot.ext | character, default: ".pdf". Alternatives: ".png" or ".jpg". Note that in pdf format the complex correlation plots become very slow to load! |
| create.ofst | logical, default TRUE. Create ".ofst" files from the input libraries, ofst is much faster to load in R, for later use. Stored in ./ofst/ folder relative to experiment main folder. |
| complex.correlation.plots | logical, default TRUE. Add in addition to simple correlation plot two computationally heavy dots + correlation plots. Useful for deeper analysis, but takes longer time to run, especially on low-quality gpu computers. Set to FALSE to skip these. |
| library.names | character, default: bamVarName(df). Names to load libraries as to environment and names to display in plots. |
| use_simplified_reads | logical, default TRUE. For count tables and coverage plots a speed up for GAlignments is to use 5' ends only. This will lose some detail for splice sites, but is usually irrelevant. Note: If reads are precollapsed GRanges, set to FALSE to avoid recollapsing. |
| BPPARAM | how many cores/threads to use? default: bpparam(). To see number of threads used, do bpparam()$workers. You can also add a time remaining bar, for a more detailed pipeline. |

## Value

invisible(NULL) (objects are stored to disc)

## See Also

Other QC report: QCplots(), QCstats()

## Examples

```
# Load an experiment
df <- ORFik.template.experiment()
# Run QC
#QCreport(df, tempdir())
# QC on subset
#QCreport(df[9,], tempdir())
```

## QCstats *Load ORFik QC Statistics report*

### Description

Loads the pre / post alignment statistcs made in ORFik.

### Usage

```
QCstats(df, path = file.path(QCfolder(df), "STATS.csv"))
```

### Arguments

| | |
|---|---|
| df | an ORFik experiment |
| path | path to QC statistics report, default: file.path(dirname(df$filepath[1]), "/QC_STATS/STATS.csv") |

### Details

The ORFik QC uses the aligned files (usually bam files), fastp and STAR log files combined with annotation to create relevant statistics.

### Value

data.table of QC report or NULL if not exists

### See Also

Other QC report: QCplots(), QCreport()

### Examples

```
df <- ORFik.template.experiment()
## First make QC report
# QCreport(df)
# stats <- QCstats(df)
```

## QCstats.plot *Make plot of ORFik QCreport*

### Description

From post-alignment QC relative to annotation, make a plot for all samples. Will contain among others read lengths, reads overlapping leaders, cds, trailers, mRNA / rRNA etc.

### Usage

```
QCstats.plot(stats, output.dir = NULL, plot.ext = ".pdf", as_gg_list = FALSE)
```

### Arguments

| | |
|---|---|
| stats | the experiment object or path to custom ORFik QC folder where a file called "STATS.csv" is located. |
| output.dir | NULL or character path, default: NULL, plot not saved to disc. If defined saves plot to that directory with the name "/STATS_plot.pdf". |
| plot.ext | character, default: ".pdf". Alternatives: ".png" or ".jpg". |
| as_gg_list | logical, default FALSE. Return as a list of ggplot objects instead of as a grob. Gives you the ability to modify plots more directly. |

### Value

the plot object, a grob of ggplot objects of the the statistics data

### Examples

```
df <- ORFik.template.experiment()[3,]
## First make QC report
# QCreport(df)
## Now you can get plot
# QCstats.plot(df)
```

---

## r *strandMode covRle*

---

### Description

strandMode covRle

### Usage

```
r(x)
```

### Arguments

| | |
|---|---|
| x | a covRle object |

### Value

the forward RleList

## r,covRle-method *strandMode covRle*

### Description

strandMode covRle

### Usage

```
## S4 method for signature 'covRle'
r(x)
```

### Arguments

| | |
|---|---|
| x | a covRle object |

### Value

the forward RleList

## rankOrder *ORF rank in transcripts*

### Description

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

### Usage

```
rankOrder(grl)
```

### Arguments

| | |
|---|---|
| grl | a GRangesList object with ORFs |

### Value

a numeric vector of integers

### References

doi: 10.1074/jbc.R116.733899

## See Also

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

## Examples

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

---

# read.experiment *Read ORFik experiment*

---

## Description

Read in runs / samples from an experiment as a single R object. To read an ORFik experiment, you must of course make one first. See create.experiment The file must be csv and be a valid ORFik experiment

## Usage

```
read.experiment(
  file,
  in.dir = ORFik::config()["exp"],
  validate = TRUE,
  output.env = .GlobalEnv
)
```

## Arguments

| | |
|---|---|
| file | relative path to a ORFik experiment. That is a .csv file following ORFik experiment style ("," as seperator). , or a template data.frame from create.experiment. Can also be full path to file, then in.dir argument is ignored. |
| in.dir | Directory to load experiment csv file from, default: ORFik::config()["exp"], which has default "~/Bio_data/ORFik_experiments/" Set to NULL if you don't want to save it to disc. Does not apply if file argument is not a path (can also be a data.frame). Also does not apply if file argument was given as full path. |

| | |
|---|---|
| validate | logical, default TRUE. Abort if any library files does not exist. Do not set this to FALSE, unless you know what you are doing! |
| output.env | an environment, default .GlobalEnv. Which environment should ORFik output libraries to (if this is done), can be updated later with envExp(df) <- new.env(). |

## Value

an ORFik experiment

## See Also

Other ORFik_experiment: ORFik.template.experiment(), ORFik.template.experiment.zf(), bamVarName(), create.experiment(), experiment-class, filepath(), libraryTypes(), organism,experiment-meth outputLibs(), save.experiment(), validateExperiments()

## Examples

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

## End(Not run)
## Read from (create.experiment() template)
df <- ORFik.template.experiment()

## To save it, do:
# save.experiment(df, file = "path/to/save/experiment")
## You can then do:
# read.experiment("path/to/save/experiment")
# or (identical):
# read.experiment("experiment", in.dir = "path/to/save/")
```

---

# readBam *Custom bam reader*

---

## Description

Read in Bam file from either single end or paired end. Safer combined version of readGAlignments and readGAlignmentPairs that takes care of some common errors.
If QNAMES of the aligned reads are from collapsed fasta files (if the names are formated from collapsing in either (ORFik, ribotoolkit or fastx)), the bam file will contain a meta column called "score" with the counts of duplicates per read. Only works for single end reads, as perfect duplication events for paired end is more rare and therefor not supported!.

## Usage

```
readBam(path, chrStyle = NULL, param = NULL, strandMode = 0)
```

## Arguments

| | |
|---|---|
| path | a character / data.table with path to .bam file. There are 3 input file possibilities.<br>• single end : a character path (length 1)<br>• paired end (1 file) : Either a character path (length of 2), where path[2] is "paired-end", or a data.table with 2 columns, forward = path & reverse = "paired-end"<br>• paired end (2 files) : Either a character path (length of 2), where path[2] is path to R2, or a data.table with 2 columns, forward = path to R1 & reverse = path to R2. (This one is not used often) |
| chrStyle | a GRanges object, TxDb, FaFile, , a seqlevelsStyle or Seqinfo. (Default: NULL) to get seqlevelsStyle from. In addition if it is a Seqinfo object, seqinfo will be updated. Example of seqlevelsStyle update: Is chromosome 1 called chr1 or 1, is mitocondrial chromosome called MT or chrM etc. Will use 1st seqlevel-style if more are present. Like: c("NCBI", "UCSC") -> pick "NCBI" |
| param | NULL or a ScanBamParam object. Like for scanBam, this influences what fields and which records are imported. However, note that the fields specified thru this ScanBamParam object will be loaded *in addition* to any field required for generating the returned object (GAlignments, GAlignmentPairs, or GappedReads object), but only the fields requested by the user will actually be kept as metadata columns of the object.<br>By default (i.e. param=NULL or param=ScanBamParam()), no additional field is loaded. The flag used is scanBamFlag(isUnmappedQuery=FALSE) for readGAlignments, readGAlignmentsList, and readGappedReads. (i.e. only records corresponding to mapped reads are loaded), and scanBamFlag(isUnmappedQuery=FALSE, isPaired=TRUE, hasUnmappedMate=FALSE) for readGAlignmentPairs (i.e. only records corresponding to paired-end reads with both ends mapped are loaded). |
| strandMode | numeric, default 0. Only used for paired end bam files. One of (0: strand = *, 1: first read of pair is +, 2: first read of pair is -). See ?strandMode. Note: Sets default to 0 instead of 1, as readGAlignmentPairs uses 1. This is to guarantee hits, but will also make mismatches of overlapping transcripts in opposite directions. |

## Details

In the future will use a faster .bam loader for big .bam files in R.

## Value

a GAlignments or GAlignmentPairs object of bam file

## See Also

Other utils: bedToGR(), convertToOneBasedRanges(), export.bed12(), export.bigWig(), export.fstwig(), export.wiggle(), fimport(), findFa(), fread.bed(), optimizeReads(), readBigWig(), readWig()

## Examples

```
bam_file <- system.file("extdata/Danio_rerio_sample", "ribo-seq.bam", package = "ORFik")
readBam(bam_file, "UCSC")
```

---

`readBigWig` *Custom bigWig reader*

---

## Description

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

## Usage

```
readBigWig(path, chrStyle = NULL, as = "GRanges")
```

## Arguments

| | |
|---|---|
| `path` | a character path to two .bigWig files, or a data.table with 2 columns, (forward, filepath) and reverse, only 1 row. |
| `chrStyle` | a GRanges object, TxDb, FaFile, , a `seqlevelsStyle` or `Seqinfo`. (Default: NULL) to get seqlevelsStyle from. In addition if it is a Seqinfo object, seqinfo will be updated. Example of seqlevelsStyle update: Is chromosome 1 called chr1 or 1, is mitocondrial chromosome called MT or chrM etc. Will use 1st seqlevel-style if more are present. Like: c("NCBI", "UCSC") -> pick "NCBI" |
| `as` | Specifies the class of the return object. Default is `GRanges`, which has one range per range in the file, and a score column holding the value for each range. For `NumericList`, one numeric vector is returned for each range in the `selection` argument. For `RleList`, there is one `Rle` per sequence, and that `Rle` spans the entire sequence. |

## Value

a `GRanges` object of the file/s

## See Also

Other utils: `bedToGR()`, `convertToOneBasedRanges()`, `export.bed12()`, `export.bigWig()`, `export.fstwig()`, `export.wiggle()`, `fimport()`, `findFa()`, `fread.bed()`, `optimizeReads()`, `readBam()`, `readWig()`

readWidths *Get read widths*

## Description

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

## Usage

```
readWidths(reads, after.softclips = TRUE, along.reference = FALSE)
```

## Arguments

| | |
|---|---|
| reads | a GRanges, GAlignment or GAlignmentPairs object. |
| after.softclips | logical (TRUE), include softclips in width. Does not apply if along.reference is TRUE. |
| along.reference | logical (FALSE), example: The cigar "26MI2" is by default width 28, but if along.reference is TRUE, it will be 26. The length of the read along the reference. Also "1D20M" will be 21 if by along.reference is TRUE. Intronic regions (cigar: N) will be removed. So: "1M200N19M" is 20, not 220. |

## Details

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

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

## Value

an integer vector of widths

## Examples

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

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

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

---

`readWig` *Custom wig reader*

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| `path` | a character path to two .wig files, or a data.table with 2 columns, (forward, filepath) and reverse, only 1 row. |
| `chrStyle` | a GRanges object, TxDb, FaFile, , a `seqlevelsStyle` or `Seqinfo`. (Default: NULL) to get seqlevelsStyle from. In addition if it is a Seqinfo object, seqinfo will be updated. Example of seqlevelsStyle update: Is chromosome 1 called chr1 or 1, is mitocondrial chromosome called MT or chrM etc. Will use 1st seqlevel-style if more are present. Like: c("NCBI", "UCSC") -> pick "NCBI" |

## Value

a `GRanges` object of the file/s

## See Also

Other utils: `bedToGR()`, `convertToOneBasedRanges()`, `export.bed12()`, `export.bigWig()`, `export.fstwig()`, `export.wiggle()`, `fimport()`, `findFa()`, `fread.bed()`, `optimizeReads()`, `readBam()`, `readBigWig()`

---

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

---

## Description

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

## Usage

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

## Arguments

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

## Details

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

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

## Value

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

## See Also

Other CAGE: `assignTSSByCage()`, `reassignTxDbByCage()`

## Examples

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

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

---

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

---

## Description

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

## Usage

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

```
  removeUnused = FALSE,
  preCleanup = TRUE
)
```

## Arguments

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

## Details

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

## Value

a TxDb obect of reassigned transcripts

## See Also

Other CAGE: `assignTSSByCage()`, `reassignTSSbyCage()`

## Examples

```
## Not run:
library(GenomicFeatures)
# Get the gtf txdb file
txdbFile <- system.file("extdata", "hg19_knownGene_sample.sqlite",
```

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

## End(Not run)
```

---

`reduceKeepAttr` *Reduce GRanges / GRangesList*

---

## Description

Reduce away all GRanges elements with 0-width.

## Usage

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

## 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 between two ranges, to merge them. |
| `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. |
| `min.strand.decreasing` | (TRUE), if GRangesList, return minus strand group ranges in decreasing order (1-5, 30-50) -> (30-50, 1-5) |

## Details

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

## Value

A reduced GRangesList

## See Also

Other ExtendGenomicRanges: asTX(), coveragePerTiling(), extendLeaders(), extendTrailers(), 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)
```

---

# regionPerReadLength *Find proportion of reads per position per read length in region*

---

## Description

This is defined as: Given some transcript region (like CDS), get coverage per position. By default only returns positions that have hits, set drop.zero.dt to FALSE to get all 0 positions.

## Usage

```
regionPerReadLength(
  grl,
  reads,
  acceptedLengths = NULL,
  withFrames = TRUE,
  scoring = "transcriptNormalized",
  weight = "score",
  exclude.zero.cov.grl = TRUE,
  drop.zero.dt = TRUE,
  BPPARAM = bpparam()
)
```

## Arguments

| | |
|---|---|
| grl | a GRangesList object with usually either leaders, cds', 3' utrs or ORFs |

reads — a GAlignments, GRanges, or precomputed coverage as covRleList (where names of covRle objects are readlengths) of RiboSeq, RnaSeq etc.
Weigths for scoring is default the 'score' column in 'reads'. Can also be random access paths to bigWig or fstwig file. Do not use random access for more than a few genes, then loading the entire files is usually better.

acceptedLengths — an integer vector (NULL), the read lengths accepted. Default NULL, means all lengths accepted.

withFrames — logical TRUE, add ORF frame (frame 0, 1, 2), starting on first position of every grl.

scoring — a character (transcriptNormalized), which meta coverage scoring ? one of (zscore, transcriptNormalized, mean, median, sum, sumLength, fracPos), see ?coverageScorings for more info. Use to decide a scoring of hits per position for metacoverage etc. Set to NULL if you do not want meta coverage, but instead want per gene per position raw counts.

weight — (default: 'score'), if defined a character name of valid meta column in subject. GRanges("chr1", 1, "+", score = 5), would mean score column tells that this alignment region was found 5 times. ORFik ofst, bedoc and .bedo files contains a score column like this. As do CAGEr CAGE files and many other package formats. You can also assign a score column manually.

exclude.zero.cov.grl — logical, default TRUE. Do not include ranges that does not have any coverage (0 reads on them), this makes it faster to run.

drop.zero.dt — logical, default TRUE. If TRUE and as.data.table is TRUE, remove all 0 count positions. This greatly speeds up and most importantly, greatly reduces memory usage. Will not change any plots, unless 0 count positions are used in some sense.

BPPARAM — how many cores/threads to use? default: bpparam()

## Value

a data.table with lengths by coverage.

## See Also

Other coverage: coverageScorings(), metaWindow(), scaledWindowPositions(), windowPerReadLength()

## Examples

```
# Raw counts per gene per position
cds <- GRangesList(tx1 = GRanges("1", 100:129, "+"))
reads <- GRanges("1", seq(79,129, 3), "+")
reads$size <- 28 # <- Set read length of reads
regionPerReadLength(cds, reads, scoring = NULL)
## Sum up reads in each frame per read length per gene
regionPerReadLength(cds, reads, scoring = "frameSumPerLG")
```

## remove.experiments *Remove ORFik experiment libraries load in R*

### Description

Variable names defined by df, in envir defined

### Usage

```
remove.experiments(df, envir = envExp(df))
```

### Arguments

| | |
|---|---|
| df | an ORFik experiment |
| envir | environment to save to, default envExp(df), which defaults to .GlobalEnv, but can be set with envExp(df) <- new.env() etc. |

### Value

NULL (objects removed from envir specified)

### Examples

```
df <- ORFik.template.experiment()
# Output to .GlobalEnv with:
# outputLibs(df)
# Then remove them with:
# remove.experiments(df)
```

## resFolder *Get ORFik experiment main output directory*

### Description

Get ORFik experiment main output directory

### Usage

```
resFolder(x)
```

### Arguments

| | |
|---|---|
| x | an ORFik experiment |

### Value

a character path

---

`resFolder,experiment-method` *Get ORFik experiment main output directory*

---

## Description

Get ORFik experiment main output directory

## Usage

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

## Arguments

| | |
|---|---|
| `x` | an ORFik `experiment` |

## Value

a character path

---

`riboORFs` *Load Predicted translons*

---

## Description

Load Predicted translons

## Usage

```
riboORFs(df, type = "table", folder = riboORFsFolder(df))
```

## Arguments

| | |
|---|---|
| `df` | ORFik experiment |
| `type` | default "table", alternatives: c("table", "ranges_candidates", "ranges_predictions", "predictions") |
| `folder` | base folder to check for computed results, default: riboORFsFolder(df) |

## Value

a data.table, GRangesList or list of logical vector depending on input

## Examples

```
df <- ORFik.template.experiment()
df <- df[df$libtype == "RFP",][c(1,2),]
# riboORFs(df) # Works when you have run prediction
```

## riboORFsFolder *Define folder for prediction output*

### Description

Define folder for prediction output

### Usage

```
riboORFsFolder(df, parrent_dir = resFolder(df))
```

### Arguments

| | |
|---|---|
| df | ORFik experiment |
| parrent_dir | Parrent directory of computed study results, default: resFolder(df) |

### Value

a file path (full path)

### Examples

```
df <- ORFik.template.experiment()
df <- df[df$libtype == "RFP",][c(1,2),]
riboORFsFolder(df)
riboORFsFolder(df, tempdir())
```

## RiboQC.plot *Quality control for pshifted Ribo-seq data*

### Description

Combines several statistics from the pshifted reads into a plot:
-1 Coding frame distribution per read length
-2 Alignment statistics
-3 Biotype of non-exonic pshifted reads
-4 mRNA localization of pshifted reads

## Usage

```
RiboQC.plot(
  df,
  output.dir = QCfolder(df),
  width = 6.6,
  height = 4.5,
  plot.ext = ".pdf",
  type = "pshifted",
  weight = "score",
  bar.position = "dodge",
  as_gg_list = FALSE,
  BPPARAM = BiocParallel::SerialParam(progressbar = TRUE)
)
```

## Arguments

| | |
|---|---|
| df | an ORFik experiment |
| output.dir | NULL or character path, default: NULL, plot not saved to disc. If defined saves plot to that directory with the name "/STATS_plot.pdf". |
| width | width of plot, default 6.6 (in inches) |
| height | height of plot, default 4.5 (in inches) |
| plot.ext | character, default: ".pdf". Alternatives: ".png" or ".jpg". |
| type | type of library loaded, default pshifted, warning if not pshifted might crash if too many read lengths! |
| weight | which column in reads describe duplicates, default "score". |
| bar.position | character, default "dodge". Should Ribo-seq frames per read length be positioned as "dodge" or "stack" (on top of each other). |
| as_gg_list | logical, default FALSE. Return as a list of ggplot objects instead of as a grob. Gives you the ability to modify plots more directly. |
| BPPARAM | how many cores/threads to use? default: bpparam(). To see number of threads used, do bpparam()$workers. You can also add a time remaining bar, for a more detailed pipeline. |

## Value

the plot object, a grob of ggplot objects of the the data

## Examples

```
df <- ORFik.template.experiment()
df <- df[9,] #lets only p-shift RFP sample at index 9
#shiftFootprintsByExperiment(df)
#RiboQC.plot(df, tempdir())
```

`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,
  weight.RFP = 1L,
  weight.RNA = 1L,
  overlapGrl = NULL
)
```

## Arguments

| | |
|---|---|
| `grl` | a `GRangesList` object with usually either leaders, cds', 3' utrs or ORFs. |
| `RFP` | RiboSeq reads as GAlignments, 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 `GAlignments`, `GRanges` or GRangesList object |
| `weight.RFP` | a vector (default: 1L). Can also be character name of column in RFP. As in translationalEff(weight = "score") for: GRanges("chr1", 1, "+", score = 5), would mean score column tells that this alignment region was found 5 times. |
| `weight.RNA` | Same as weightRFP but for RNA weights. (default: 1L) |
| `overlapGrl` | an integer, (default: NULL), if defined must be countOverlaps(grl, RFP), added for speed if you already have it |

## 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: computeFeatures(), computeFeaturesCage(), countOverlapsW(), disengagementScore(), distToCds(), distToTSS(), entropy(), floss(), fpkm(), fpkm_calc(), fractionLength(), initiationScore(), insideOutsideORF(), isInFrame(), isOverlapping(), kozakSequenceScore(), orfScore(), rankOrder(), ribosomeStallingScore(), startRegion(), startRegionCoverage(), stopRegion(), 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, weight = 1L, overlapGrl = NULL)
```

## Arguments

| | |
|---|---|
| grl | a GRangesList object with usually either leaders, cds', 3' utrs or ORFs. |
| RFP | RiboSeq reads as GAlignments, GRanges or GRangesList object |
| weight | a vector (default: 1L, if 1L it is identical to countOverlaps()), if single number (!= 1), it applies for all, if more than one must be equal size of 'reads'. else it must be the string name of a defined meta column in subject "reads", that gives number of times a read was found. GRanges("chr1", 1, "+", score = 5), would mean "score" column tells that this alignment region was found 5 times. |
| overlapGrl | an integer, (default: NULL), if defined must be countOverlaps(grl, RFP), added for speed if you already have it |

## Value

a named vector of numeric values of RSS scores

## References

doi: 10.1016/j.cels.2017.08.004

## See Also

Other features: computeFeatures(), computeFeaturesCage(), countOverlapsW(), disengagementScore(), distToCds(), distToTSS(), entropy(), floss(), fpkm(), fpkm_calc(), fractionLength(), initiationScore(), insideOutsideORF(), isInFrame(), isOverlapping(), kozakSequenceScore(), orfScore(), rankOrder(), ribosomeReleaseScore(), startRegion(), startRegionCoverage(), stopRegion(), 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)
```

---

# ribo_fft *Get periodogram data per read length*

---

## Description

A data.table of periods and amplitudes, great to detect ribosomal read lengths. Uses 5' end of reads to detect periodicity. Works both before and after p-shifting. Plot results with ribo_fft_plot.

## Usage

```
ribo_fft(footprints, cds, read_lengths = 26:34, firstN = 150)
```

## Arguments

| | |
|---|---|
| footprints | Ribosome footprints in either GAlignments or GRanges |
| cds | a GRangesList of coding sequences. Length must match length of argument mrna, and all must have length > arugment firstN. |
| read_lengths | integer vector, default: 26:34, which read length to check for. Will exclude all read_lengths that does not exist for footprints. |
| firstN | (integer) Represents how many bases of the transcripts downstream of start codons to use for initial estimation of the periodicity. |

## Value

a data.table with read_length, amplitude and periods

## Examples

```
## Note, this sample data is not intended to be strongly periodic.
## Real data should have a cleaner peak for x = 3 (periodicity)
# Load sample data
df <- ORFik.template.experiment()
# Load annotation
loadRegions(df, "cds", names.keep = filterTranscripts(df))
# Select a riboseq library
df <- df[df$libtype == "RFP", ]
footprints <- fimport(filepath(df[1,], "default"))
fft_dt <-ribo_fft(footprints, cds)
ribo_fft_plot(fft_dt)
```

---

# ribo_fft_plot *Get periodogram plot per read length*

---

## Description

Get periodogram plot per read length

## Usage

```
ribo_fft_plot(fft_dt, period_window = c(0, 6))
```

## Arguments

| | |
|---|---|
| fft_dt | a data.table with read_length, amplitude and periods |
| period_window | x axis limits, default c(0,6) |

## Value

a ggplot, geom_line plot facet by read length.

## Examples

```
## Note, this sample data is not intended to be strongly periodic.
## Real data should have a cleaner peak for x = 3 (periodicity)
# Load sample data
df <- ORFik.template.experiment()
# Load annotation
cds <- loadRegion(df, "cds", names.keep = filterTranscripts(df))
# Select a riboseq library
df <- df[df$libtype == "RFP", ]
footprints <- fimport(filepath(df[1,], "default"))
fft_dt <-ribo_fft(footprints, cds)
ribo_fft_plot(fft_dt)
```

## rnaNormalize *Normalize a data.table of coverage by RNA seq per position*

### Description

Normalizes per position per gene by this function: (reads at position / min(librarysize, 1) * number of genes) / fpkm of that gene's RNA-seq

### Usage

```
rnaNormalize(coverage, df, dfr = NULL, tx, normalizeMode = "position")
```

### Arguments

| | |
|---|---|
| coverage | a data.table containing at least columns (count/score, position), it is possible to have additionals: (genes, fraction, feature) |
| df | an ORFik experiment |
| dfr | an ORFik experiment of RNA-seq to normalize against. Will add RNA normalized to plot name if this is done. |
| tx | a GRangesList of mrna transcripts |
| normalizeMode | a character (default: "position"), how to normalize library against rna library. Either on "position", normalize by number of genes, sum of reads and RNA seq, on tx "region" or "feature": same as position but RNA is split into the feature groups to normalize. Useful if you have a list of targets and background genes. |

### Details

Good way to compare libraries

### Value

a data.table of normalized transcripts by RNA.

## runIDs *Get SRR/DRR/ERR run ids from ORFik experiment*

### Description

Get SRR/DRR/ERR run ids from ORFik experiment

### Usage

```
runIDs(x)
```

### Arguments

| | |
|---|---|
| x | an ORFik experiment |

### Value

a character vector of runIDs, "" if not existing.

---

## runIDs,experiment-method *Get SRR/DRR/ERR run ids from ORFik experiment*

---

### Description

Get SRR/DRR/ERR run ids from ORFik experiment

### Usage

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

### Arguments

| | |
|---|---|
| x | an ORFik experiment |

### Value

a character vector of runIDs, "" if not existing.

---

## save.experiment *Save experiment to disc*

---

### Description

Save experiment to disc

### Usage

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

### Arguments

| | |
|---|---|
| df | an ORFik experiment |
| file | name of file to save df as |

## Value

NULL (experiment save only)

## See Also

Other ORFik_experiment: ORFik.template.experiment(), ORFik.template.experiment.zf(), bamVarName(), create.experiment(), experiment-class, filepath(), libraryTypes(), organism,experiment-meth outputLibs(), read.experiment(), validateExperiments()

## Examples

```
df <- ORFik.template.experiment()
## Save with:
#save.experiment(df, file = "path/to/save/experiment.csv")
## Identical (.csv not needed, can be added):
#save.experiment(df, file = "path/to/save/experiment")
```

---

# scaledWindowPositions *Scale (bin) windows to a meta window of given size*

---

## Description

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

## Usage

```
scaledWindowPositions(
  grl,
  reads,
  scaleTo = 100,
  scoring = "meanPos",
  weight = "score",
  is.sorted = FALSE,
  drop.zero.dt = FALSE
)
```

## Arguments

| | |
|---|---|
| grl | a GRangesList of 5' utrs, CDS, transcripts, etc. |
| reads | a GAlignments, GRanges, or precomputed coverage as covRle (one for each strand) of RiboSeq, RnaSeq etc.<br>Weigths for scoring is default the 'score' column in 'reads'. Can also be random access paths to bigWig or fstwig file. Do not use random access for more than a few genes, then loading the entire files is usually better. File streaming is still in beta, so use with care! |

| | |
|---|---|
| scaleTo | an integer (100), if windows have different size, a meta window can not directly be created, since a meta window must have equal size for all windows. Rescale all windows to scaleTo. i.e c(1,2,3) -> size 2 -> c(1, mean(2,3)) etc. Can also be a vector, 1 number per grl group. |
| scoring | a character, one of (meanPos, sumPos, ..) Check the coverageScoring function for more options. |
| weight | (default: 'score'), if defined a character name of valid meta column in subject. GRanges("chr1", 1, "+", score = 5), would mean score column tells that this alignment region was found 5 times. ORFik ofst, bedoc and .bedo files contains a score column like this. As do CAGEr CAGE files and many other package formats. You can also assign a score column manually. |
| is.sorted | logical (FALSE), is grl sorted. That is + strand groups in increasing ranges (1,2,3), and - strand groups in decreasing ranges (3,2,1) |
| drop.zero.dt | logical FALSE, if TRUE and as.data.table is TRUE, remove all 0 count positions. This greatly speeds up and most importantly, greatly reduces memory usage. Will not change any plots, unless 0 positions are used in some sense. (mean, median, zscore coverage will only scale differently) |

## Details

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

## Value

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

## See Also

Other coverage: coverageScorings(), metaWindow(), regionPerReadLength(), windowPerReadLength()

## Examples

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

## scoreSummarizedExperiment *Helper function for makeSummarizedExperimentFromBam*

### Description

If txdb or gtf path is added, it is a rangedSummerizedExperiment For FPKM values, DESeq2::fpkm(robust = FALSE) is used

### Usage

```
scoreSummarizedExperiment(
  final,
  score = "transcriptNormalized",
  collapse = FALSE
)
```

### Arguments

| | |
|---|---|
| final | ranged summarized experiment object |
| score | default: "transcriptNormalized" (row normalized raw counts matrix), alternative is "fpkm", "log2fpkm" or "log10fpkm" |
| collapse | a logical/character (default FALSE), if TRUE all samples within the group SAMPLE will be collapsed to one. If "all", all groups will be merged into 1 column called merged_all. Collapse is defined as rowSum(elements_per_group) / ncol(elements_per_group) |

### Value

a DEseq summerizedExperiment object (transcriptNormalized) or matrix (if fpkm input)

## seqinfo,covRle-method *Seqinfo covRle Extracted from forward RleList*

### Description

Seqinfo covRle Extracted from forward RleList

### Usage

```
## S4 method for signature 'covRle'
seqinfo(x)
```

### Arguments

| | |
|---|---|
| x | a covRle object |

### Value

integer vector with names

---

`seqinfo,covRleList-method`

*Seqinfo covRle Extracted from forward RleList*

---

### Description

Seqinfo covRle Extracted from forward RleList

### Usage

```
## S4 method for signature 'covRleList'
seqinfo(x)
```

### Arguments

| | |
|---|---|
| x | a covRle object |

### Value

integer vector with names

---

`seqinfo,experiment-method`

*Seqinfo ORFik experiment Extracted from fasta genome index*

---

### Description

Seqinfo ORFik experiment Extracted from fasta genome index

### Usage

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

### Arguments

| | |
|---|---|
| x | an ORFik `experiment` |

### Value

integer vector with names

seqlevels,covRle-method
*Seqlevels covRle Extracted from forward RleList*

## Description

Seqlevels covRle Extracted from forward RleList

## Usage

```
## S4 method for signature 'covRle'
seqlevels(x)
```

## Arguments

| | |
|---|---|
| x | a covRle object |

## Value

integer vector with names

seqlevels,covRleList-method
*Seqlevels covRleList Extracted from forward RleList*

## Description

Seqlevels covRleList Extracted from forward RleList

## Usage

```
## S4 method for signature 'covRleList'
seqlevels(x)
```

## Arguments

| | |
|---|---|
| x | a covRle object |

## Value

integer vector with names

---

`seqlevels,experiment-method` *Seqlevels ORFik experiment Extracted from fasta genome index*

---

## Description

Seqlevels ORFik experiment Extracted from fasta genome index

## Usage

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

## Arguments

| | |
|---|---|
| x | an ORFik experiment |

## Value

integer vector with names

---

`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, default: (TRUE) |

## 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 and soft clips 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 ofst, covRle, bed or wig file.

## Usage

```
shiftFootprints(footprints, shifts, sort = TRUE)
```

## Arguments

| | |
|---|---|
| `footprints` | `GAlignments` object of RiboSeq reads |
| `shifts` | a data.frame / data.table with minimum 2 columns, fraction (selected read lengths) and offsets_start (relative position in nt). Output from `detectRibosomeShifts`. Run `ORFik::shifts.load(df)[[1]]` for an example of input. |
| `sort` | logical, default TRUE. If False will keep original order of reads, and not sort output reads in increasing genomic location per chromosome and strand. |

## Details

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

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

## Value

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

## References

https://bmcgenomics.biomedcentral.com/articles/10.1186/s12864-018-4912-6

## See Also

Other pshifting: changePointAnalysis(), detectRibosomeShifts(), shiftFootprintsByExperiment(), shiftPlots(), shifts_load(), shifts_save()

## Examples

```
## Basic run
# Transcriptome annotation ->
gtf_file <- system.file("extdata/references/danio_rerio", "annotations.gtf", package = "ORFik")
# Ribo seq data ->
riboSeq_file <- system.file("extdata/Danio_rerio_sample", "ribo-seq.bam", package = "ORFik")
## Not run:
footprints <- readBam(riboSeq_file)

# detect the shifts automagically
shifts <- detectRibosomeShifts(footprints, gtf_file)
# shift the RiboSeq footprints
shiftedReads <- shiftFootprints(footprints, shifts)

## End(Not run)
```

---

# shiftFootprintsByExperiment

*Shift footprints of each file in experiment*

---

## Description

A function that combines the steps of periodic read length detection, p-site shift detection and p-shifting into 1 function. For more details, see: detectRibosomeShifts
Saves files to a specified location as .ofst and .wig, The .ofst file will include a score column containing read width.
The .wig files, will be saved in pairs of +/- strand, and score column will be replicates of reads starting at that position, score = 5 means 5 reads.
Remember that different species might have different default Ribosome read lengths, for human, mouse etc, normally around 27:30.

## Usage

```
shiftFootprintsByExperiment(
  df,
  out.dir = pasteDir(libFolder(df), "/pshifted/"),
  start = TRUE,
  stop = FALSE,
  top_tx = 10L,
  minFiveUTR = 30L,
  minCDS = 150L,
  minThreeUTR = if (stop) {
     30
 } else NULL,
  firstN = 150L,
  min_reads = 1000,
  min_reads_TIS = 50,
  accepted.lengths = 26:34,
  output_format = c("ofst", "wig"),
  BPPARAM = bpparam(),
  tx = NULL,
  shift.list = NULL,
  log = TRUE,
  heatmap = FALSE,
  must.be.periodic = TRUE,
  strict.fft = TRUE,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| df | an ORFik experiment |
| out.dir | output directory for files, default: pasteDir(libFolder(df), "/pshifted/"), making a /pshifted folder inside default bam file location |
| start | (logical) Whether to include predictions based on the start codons. Default TRUE. |
| stop | (logical) Whether to include predictions based on the stop codons. Default FASLE. Only use if there exists 3' UTRs for the annotation. If peridicity around stop codon is stronger than at the start codon, use stop instead of start region for p-shifting. |
| top_tx | (integer), default 10. Specify which % of the top TIS coverage transcripts to use for estimation of the shifts. By default we take top 10 top covered transcripts as they represent less noisy data-set. This is only applicable when there are more than 1000 transcripts. |
| minFiveUTR | (integer) minimum bp for 5' UTR during filtering for the transcripts. Set to NULL if no 5' UTRs exists for annotation. |
| minCDS | (integer) minimum bp for CDS during filtering for the transcripts |
| minThreeUTR | (integer) minimum bp for 3' UTR during filtering for the transcripts. Set to NULL if no 3' UTRs exists for annotation. |

`firstN` (integer) Represents how many bases of the transcripts downstream of start codons to use for initial estimation of the periodicity.

`min_reads` default (1000), how many reads must a read-length have in total to be considered for periodicity.

`min_reads_TIS` default (50), how many reads must a read-length have in the TIS region to be considered for periodicity.

`accepted.lengths` accepted read lengths, default 26:34, usually ribo-seq is strongest between 27:32.

`output_format` default c("ofst", "wig"), use export.ofst or wiggle format (wig) using `export.wiggle` ? Default is both.
Options are: c("ofst", "bigWig", "wig", "bed", "bedo") For future coverage per nucleotide, we advice to do here ofst and bigWig for other genome browsers, then call `convert_to_covRleList` to get much faster R objects.
The wig format version can be used in IGV, the score column is counts of that read with that read length, the cigar reference width is lost, ofst is much faster to save and load in R, and retain cigar reference width, but can not be used in IGV. Also for larger tracks, you can use "bigWig".

`BPPARAM` how many cores/threads to use? default: bpparam()

`tx` a GRangesList, if you do not have 5' UTRs in annotation, send your own version. Example: extendLeaders(tx, 30) Where 30 bases will be new "leaders". Since each original transcript was either only CDS or non-coding (filtered out).

`shift.list` default NULL, or a list containing named data.frames / data.tables with minimum 2 columns, fraction (selected read lengths) and offsets_start (relative position in nt). 1 named data.frame / data.table per library. Output from `detectRibosomeShifts`. Run `ORFik::shifts.load(df)` for an example of input. The names of the list must be the file.paths of the Ribo-seq libraries. Use this to edit the shifts, if you suspect some of them are wrong in an experiment.

`log` logical, default (TRUE), output a log file with parameters used and a .rds file with all shifts per library (can be loaded with `shifts.load`)

`heatmap` a logical or character string, default FALSE. If TRUE, will plot heatmap of raw reads before p-shifting to console, to see if shifts given make sense. You can also set a filepath to save the file there.

`must.be.periodic` logical TRUE, if FALSE will not filter on periodic read lengths. (The Fourier transform filter will be skipped). This is useful if you are not going to do periodicity analysis, that is: for you more coverage depth (more read lengths) is more important than only keeping the high quality periodic read lengths.

`strict.fft` logical, TRUE. Use a FFT without noise filter. This means keep only reads lengths that are "periodic for the human eye". If you want more coverage, set to FALSE, to also get read lengths that are "messy", but the noise filter detects the periodicity of 3. This should only be done when you do not need high quality periodic reads! Example would be differential translation analysis by counts over each ORF.

`verbose` logical, default FALSE. Report details of analysis/periodogram. Good if you are not sure if the analysis was correct.

## Value

NULL (Objects are saved to out.dir/pshited/"name_pshifted.ofst", wig, bedo or .bedo)

## References

https://bmcgenomics.biomedcentral.com/articles/10.1186/s12864-018-4912-6

## See Also

Other pshifting: changePointAnalysis(), detectRibosomeShifts(), shiftFootprints(), shiftPlots(), shifts_load(), shifts_save()

## Examples

```
df <- ORFik.template.experiment.zf()
df <- df[1,] #lets only p-shift first RFP sample
## Output files as both .ofst and .wig(can be viewed in IGV/UCSC)
shiftFootprintsByExperiment(df)
# If you only need in R, do: (then you get no .wig files)
#shiftFootprintsByExperiment(df, output_format = "ofst")
## With debug info:
#shiftFootprintsByExperiment(df, verbose = TRUE)
## Re-shift, if you think some are wrong
## Here as an example we update library 1, third read length to shift 12
shift.list <- shifts_load(df)
shift.list[[1]]$offsets_start[3] <- -12
#shiftFootprintsByExperiment(df, shift.list = shift.list)
## For additional speedup in R for nucleotide coverage (coveragePerTiling etc)
```

---

# shiftPlots *Plot shifted heatmaps per library*

---

## Description

Around CDS TISs, plot coverage. A good validation for you p-shifting, to see shifts are corresponding and close to the CDS TIS.

## Usage

```
shiftPlots(
  df,
  output = NULL,
  title = "Ribo-seq",
  scoring = "transcriptNormalized",
  pShifted = TRUE,
  upstream = if (pShifted) 5 else 20,
  downstream = if (pShifted) 20 else 5,
```

```
  type = "bar",
  addFracPlot = TRUE,
  plot.ext = ".pdf",
  BPPARAM = bpparam()
)
```

## Arguments

| | |
|---|---|
| df | an ORFik experiment |
| output | name to save file, full path. (Default NULL) No saving. Sett to "auto" to save to QC_STATS folder of experiment named: "pshifts_barplots.png" or "pshifts_heatmaps.png" depending on type argument. Folder must exist! |
| title | Title for top of plot, default "Ribo-seq". A more informative name could be "Ribo-seq zebrafish Chew et al. 2013" |
| scoring | which scoring scheme to use for heatmap, default "transcriptNormalized". Some alternatives: "sum", "zscore". |
| pShifted | a logical (TRUE), are Ribo-seq reads p-shifted to size 1 width reads? If upstream and downstream is set, this argument is irrelevant. So set to FALSE if this is not p-shifted Ribo-seq. |
| upstream | an integer (5), relative region to get upstream from. Default: `ifelse(!is.null(tx), ifelse(pShifted, 5, 20), min(ifelse(pShifted, 5, 20), 0))` |
| downstream | an integer (20), relative region to get downstream from. Default: `ifelse(pShifted, 20, 5)` |
| type | character, default "bar". Plot as faceted bars, gives more detailed information of read lengths, but harder to see patterns over multiple read lengths. Alternative: "heatmap", better overview of patterns over multiple read lengths. |
| addFracPlot | logical, default TRUE, add positional sum plot on top per heatmap. |
| plot.ext | default ".pdf". Alternative ".png". Only added if output is "auto". |
| BPPARAM | how many cores/threads to use? default: bpparam() |

## Value

a ggplot2 grob object

## See Also

Other pshifting: changePointAnalysis(), detectRibosomeShifts(), shiftFootprints(), shiftFootprintsByExperi
shifts_load(), shifts_save()

## Examples

```
df <- ORFik.template.experiment.zf()
df <- df[df$libtype == "RFP",][1,] #lets only p-shift first RFP sample
#shiftFootprintsByExperiment(df, output_format = "bedo)
#grob <- shiftPlots(df, title = "Ribo-seq Human ORFik et al. 2020")
#plot(grob) #Only plot in RStudio for small amount of files!
```

shifts.load *Load the shifts from experiment*

## Description

When you p-shift using the function shiftFootprintsByExperiment, you will get a list of shifts per library. To automatically load them, you can use this function. Defaults to loading pshifts, if you made a-sites or e-sites, change the path argument to ashifted/eshifted folder instead.

## Usage

```
shifts.load(
  df,
  path = file.path(libFolder(df), "pshifted", "shifting_table.rds")
)
```

## Arguments

| | |
|---|---|
| df | an ORFik experiment |
| path | path, default file.path(libFolder(df), "pshifted", "shifting_table.rds"). Path to .rds file containing the shifts as a list, one list element per shifted bam file. |

## Value

a list of the shifts, one list element per shifted bam file.

## See Also

Other pshifting: changePointAnalysis(), detectRibosomeShifts(), shiftFootprints(), shiftFootprintsByExperi
shiftPlots(), shifts_save()

## Examples

```
df <- ORFik.template.experiment()
# subset on Ribo-seq
df <- df[df$libtype == "RFP",]
#shiftFootprintsByExperiment(df)
#shifts_load(df)
```

shifts_load *Load the shifts from experiment*

## Description

When you p-shift using the function shiftFootprintsByExperiment, you will get a list of shifts per library. To automatically load them, you can use this function. Defaults to loading pshifts, if you made a-sites or e-sites, change the path argument to ashifted/eshifted folder instead.

## Usage

```
shifts_load(
  df,
  path = file.path(libFolder(df), "pshifted", "shifting_table.rds")
)
```

## Arguments

| | |
|---|---|
| df | an ORFik experiment |
| path | path, default file.path(libFolder(df), "pshifted", "shifting_table.rds"). Path to .rds file containing the shifts as a list, one list element per shifted bam file. |

## Value

a list of the shifts, one list element per shifted bam file.

## See Also

Other pshifting: changePointAnalysis(), detectRibosomeShifts(), shiftFootprints(), shiftFootprintsByExperi
shiftPlots(), shifts_save()

## Examples

```
df <- ORFik.template.experiment()
# subset on Ribo-seq
df <- df[df$libtype == "RFP",]
#shiftFootprintsByExperiment(df)
#shifts_load(df)
```

shifts_save *Save shifts for Ribo-seq*

## Description

Should be stored in pshifted folder relative to default files

## Usage

```
shifts_save(shifts, folder)
```

## Arguments

| | |
|---|---|
| shifts | a list of data.table/data.frame objects. Must be named with the full path to ofst/bam files that defines the shifts. |
| folder | directory to save file, Usually: file.path(libFolder(df), "pshifted"), where df is the ORFik experiment / or your path of default file types. It will be named file.path(folder, "shifting_table.rds"). For ORFik to work optimally, the folder should be the /pshifted/ folder relative to default files. |

## Value

invisible(NULL), file saved to disc as "shifting_table.rds".

## See Also

Other pshifting: changePointAnalysis(), detectRibosomeShifts(), shiftFootprints(), shiftFootprintsByExperi
shiftPlots(), shifts_load()

## Examples

```
df <- ORFik.template.experiment.zf()
shifts <- shifts_load(df)
original_shifts <- file.path(libFolder(df), "pshifted", "shifting_table.rds")
# Move to temp
new_shifts_path <- file.path(tempdir(), "shifting_table.rds")
new_shifts <- c(shifts, shifts)
names(new_shifts)[2] <- file.path(tempdir(), "RiboSeqTemp.ofst")
saveRDS(new_shifts, new_shifts_path)
new_shifts[[1]][1,2] <- -10
# Now update the new shifts, here we input only first
shifts_save(new_shifts[1], tempdir())
readRDS(new_shifts_path) # You still get 2 outputs
```

---

show,covRle-method *covRle show definition*

---

## Description

Show a simplified version of the covRle

## Usage

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

## Arguments

| | |
|---|---|
| object | acovRle |

## Value

print state of covRle

---

show,covRleList-method
*covRleList show definition*

---

## Description

Show a simplified version of the covRleList.

## Usage

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

## Arguments

| | |
|---|---|
| object | acovRleList |

## Value

print state of covRleList

show,experiment-method *experiment show definition*

## Description

Show a simplified version of the experiment. The show function simplifies the view so that any column of data (like replicate or stage) is not shown, if all values are identical in that column. Filepaths are also never shown.

## Usage

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

## Arguments

| | |
|---|---|
| object | an ORFik experiment |

## Value

print state of experiment

simpleLibs *Converted format of NGS libraries*

## Description

Export as either .ofst, .wig, .bigWig,.bedo (legacy format) or .bedoc (legacy format) files:
Export files as .ofst for fastest load speed into R.
Export files as .wig / bigWig for use in IGV or other genome browsers.
The input files are checked if they exist from: envExp(df).

## Usage

```
simpleLibs(
  df,
  out.dir = libFolder(df),
  addScoreColumn = TRUE,
  addSizeColumn = TRUE,
  must.overlap = NULL,
  method = "None",
  type = "ofst",
  input.type = "ofst",
```

```
  reassign.when.saving = FALSE,
  envir = envExp(df),
  force = TRUE,
  library.names = bamVarName(df),
 libs = outputLibs(df, type = input.type, chrStyle = must.overlap, library.names =
    library.names, output.mode = "list", force = force, BPPARAM = BPPARAM),
  BPPARAM = bpparam()
)
```

## Arguments

| | |
|---|---|
| df | an ORFik experiment |
| out.dir | optional output directory, default: libFolder(df), if it is NULL, it will just reassign R objects to simplified libraries. Will then create a final folder specfied as: paste0(out.dir, "/", type, "/"). Here the files will be saved in format given by the type argument. |
| addScoreColumn | logical, default TRUE, if FALSE will not add replicate numbers as score column, see ORFik::convertToOneBasedRanges. |
| addSizeColumn | logical, default TRUE, if FALSE will not add size (width) as size column, see ORFik::convertToOneBasedRanges. Does not apply for (GAlignment version of.ofst) or .bedoc. Since they contain the original cigar. |
| must.overlap | default (NULL), else a GRanges / GRangesList object, so only reads that overlap (must.overlap) are kept. This is useful when you only need the reads over transcript annotation or subset etc. |
| method | character, default "None", the method to reduce ranges, for more info see convertToOneBasedRanges |
| type | character, output format, default "ofst". Alternatives: "ofst", "bigWig", "wig","bedo" or "bedoc". Which format you want. Will make a folder within out.dir with this name containing the files. |
| input.type | character, input type "ofst". Remember this function uses the loaded libraries if existing, so this argument is usually ignored. Only used if files do not already exist. |
| reassign.when.saving | logical, default FALSE. If TRUE, will reassign library to converted form after saving. Ignored when out.dir = NULL. |
| envir | environment to save to, default envExp(df), which defaults to .GlobalEnv, but can be set with envExp(df) <- new.env() etc. |
| force | logical, default TRUE If TRUE, reload library files even if matching named variables are found in environment used by experiment (see envExp) A simple way to make sure correct libraries are always loaded. FALSE is faster if data is loaded correctly already. |
| libs | list, output of outputLibs as list of GRanges/GAlignments/GAlignmentPairs objects. Set input.type and force arguments to define parameters. |
| BPPARAM | how many cores/threads to use? default: bpparam(). To see number of threads used, do bpparam()$workers. You can also add a time remaining bar, for a more detailed pipeline. |

### Details

We advice you to not use this directly, as other function are more safe for library type conversions. See family description below. This is mostly used internally in ORFik. It is only adviced to use if large bam files are already loaded in R and conversions are wanted from those.

See `export.ofst`, `export.wiggle`, `export.bedo` and `export.bedoc` for information on file formats.
If libraries of the experiment are already loaded into environment (default: .globalEnv) is will export using those files as templates. If they are not in environment the .ofst files from the bam files are loaded (unless you are converting to .ofst then the .bam files are loaded).

### Value

invisible NULL (saves files to disc or R .GlobalEnv)

### See Also

Other lib_converters: `convert_bam_to_ofst()`, `convert_to_bigWig()`, `convert_to_covRle()`, `convert_to_covRleList()`

### Examples

```
df <- ORFik.template.experiment()
#convertLibs(df, out.dir = NULL)
# Keep only 5' ends of reads
#convertLibs(df, out.dir = NULL, method = "5prime")
```

---

## sortPerGroup *Sort a GRangesList*

---

### Description

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

### Usage

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

### Arguments

| | |
|---|---|
| `grl` | a `GRangesList` |
| `ignore.strand` | a boolean, (default FALSE): should minus strands be sorted from highest to lowest ends. If TRUE: from lowest to highest ends. |
| `quick.rev` | default: FALSE, if TRUE, given that you know all ranges are sorted from min to max for both strands, it will only reverse coordinates for minus strand groups, and only if they are in increasing order. Much quicker |

## 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)
```

---

# STAR.align.folder *Align all libraries in folder with STAR*

---

## Description

Does either all files as paired end or single end, so if you have mix, split them in two different folders.
If STAR halts at .... loading genome, it means the STAR index was aborted early, then you need to run: STAR.remove.crashed.genome(), with the genome that crashed, and rerun.

## Usage

```
STAR.align.folder(
  input.dir,
  output.dir,
  index.dir,
  star.path = STAR.install(),
  fastp = install.fastp(),
  paired.end = FALSE,
  steps = "tr-ge",
  adapter.sequence = "auto",
  quality.filtering = FALSE,
  min.length = 20,
  mismatches = 3,
  trim.front = 0,
  max.multimap = 10,
  alignment.type = "Local",
  allow.introns = TRUE,
```

```
  max.cpus = min(90, BiocParallel::bpparam()$workers),
  wait = TRUE,
  include.subfolders = "n",
  resume = NULL,
  multiQC = TRUE,
  keep.contaminants = FALSE,
  keep.unaligned.genome = FALSE,
 script.folder = system.file("STAR_Aligner", "RNA_Align_pipeline_folder.sh", package =
    "ORFik"),
 script.single = system.file("STAR_Aligner", "RNA_Align_pipeline.sh", package = "ORFik")
)
```

## Arguments

| | |
|---|---|
| input.dir | path to fast files to align, the valid input files will be search for from formats: (".fasta", ".fastq", ".fq", or ".fa") with or without compression of .gz. Also either paired end or single end reads. Pairs will automatically be detected from similarity of naming, separated by something as .1 and .2 in the end. If files are renamed, where pairs are not similarily named, this process will fail to find correct pairs! |
| output.dir | directory to save indices, default: paste0(dirname(arguments[1]), "/STAR_index/"), where arguments is the arguments input for this function. |
| index.dir | path to STAR index folder. Path returned from ORFik function STAR.index, when you created the index folders. |
| star.path | path to STAR, default: STAR.install(), if you don't have STAR installed at default location, it will install it there, set path to a runnable star if you already have it. |
| fastp | path to fastp trimmer, default: install.fastp(), if you have it somewhere else already installed, give the path. Only works for unix (linux or Mac OS), if not on unix, use your favorite trimmer and give the output files from that trimmer as input.dir here. |
| paired.end | a logical: default FALSE, alternative TRUE. If TRUE, will auto detect pairs by names. Can not be a combination of both TRUE and FALSE!<br>If running in folder mode: The folder must then contain an even number of files and they must be named with the same prefix and sufix of either _1 and _2, 1 and 2, etc. If SRR numbers are used, it will start on lowest and match with second lowest etc. |
| steps | a character, default: "tr-ge", trimming then genome alignment<br>steps of depletion and alignment wanted: The posible candidates you can use are:<br>• tr : trim reads<br>• co : contamination merged depletion<br>• ph : phix depletion<br>• rR : rrna depletion<br>• nc : ncrna depletion |

- tR : trna depletion (Mature tRNA, so no intron checks done)
- ge : genome alignment
- all: run steps: "tr-co-ge" or "tr-ph-rR-nc-tR-ge", depending on if you have merged contaminants or not

If not "all", a subset of these ("tr-co-ph-rR-nc-tR-ge")
If co (merged contaminants) is used, non of the specific contaminants can be specified, since they should be a subset of co.
The step where you align to the genome is usually always included, unless you are doing pure contaminant analysis or only trimming. For Ribo-seq and TCP(RCP-seq) you should do rR (ribosomal RNA depletion), so when you made the STAR index you need the rRNA step, either use rRNA from .gtf or manual download. (usually just download a Silva rRNA database for SSU&LSU at: https://www.arb-silva.de/) for your species.

`adapter.sequence`
: character, default: "auto". Auto detect adapter using fastp adapter auto detection, checking first 1.5M reads. (Auto detection of adapter will not work 100% of the time (if the library is of low quality), then you must rerun this function with specified adapter from fastp adapter analysis. , using FASTQC or other adapter detection tools, else alignment will most likely fail!). If already trimmed or trimming not wanted: adapter.sequence = "disable" .You can manually assign adapter like: "ATCTCGTATGCCGTCTTCTGCTTG" or "AAAAAAAAAAAAA". You can also specify one of the three presets:

  - illumina (TrueSeq ~75/100 bp sequencing): AGATCGGAAGAGC
  - small_RNA (standard for ~50 bp sequencing): TGGAATTCTCGG
  - nextera: CTGTCTCTTATA

  Paired end auto detection uses overlap sequence of pairs, to use the slower more secure paired end adapter detection, specify as: "autoPE".

`quality.filtering`
: logical, default FALSE. Not needed for modern library prep of RNA-seq, Ribo-seq etc (usually < ~ 0.5 If you are aligning bad quality data, set this to TRUE. These filters will then be applied (default of fastp), filter if:

  - Number of N bases in read: > 5
  - Read quality: > 40% of bases in the read are <Q15

`min.length`
: 20, minimum length of aligned read without mismatches to pass filter. Anything under 20 is dangerous, as chance of random hits will become high!

`mismatches`
: 3, max non matched bases. Excludes soft-clipping, this only filters reads that have defined mismatches in STAR. Only applies for genome alignment step.

`trim.front`
: 0, default trim 0 bases 5'. For Ribo-seq use default 0. Ignored if tr (trim) is not one of the arguments in "steps"

`max.multimap`
: numeric, default 10. If a read maps to more locations than specified, will skip the read. Set to 1 to only get unique mapping reads. Only applies for genome alignment step. The depletions are allowing for multimapping.

`alignment.type`
: default: "Local": standard local alignment with soft-clipping allowed, "End-ToEnd" (global): force end-to-end read alignment, does not soft-clip.

`allow.introns` logical, default TRUE. Allow large gaps of N in reads during genome alignment, if FALSE: sets –alignIntronMax to 1 (no introns). NOTE: You will still get some spliced reads if you assigned a gtf at the index step.

`max.cpus` integer, default: `min(90, BiocParallel:::bpparam()$workers)`, number of threads to use. Default is minimum of 90 and maximum cores - 2. So if you have 8 cores it will use 6. Note: FASTP will use maximum 16 threads as from testing I see performance actually degrades using anything higher. From testing I also see STAR gets no performance gain after ~50 threads. I do suspect this will change when hard drives gets better in the future.

`wait` a logical (not `NA`) indicating whether the R interpreter should wait for the command to finish, or run it asynchronously. This will be ignored (and the interpreter will always wait) if `intern = TRUE`. When running the command asynchronously, no output will be displayed on the `Rgui` console in Windows (it will be dropped, instead).

`include.subfolders` "n" (no), do recursive search downwards for fast files if "y".

`resume` default: NULL, continue from step, lets say steps are "tr-ph-ge": (trim, phix depletion, genome alignment) and resume is "ge", you will then use the assumed already trimmed and phix depleted data and start at genome alignment, useful if something crashed. Like if you specified wrong STAR version, but the trimming step was completed. Resume mode can only run 1 step at the time.

`multiQC` logical, default TRUE. Do mutliQC comparison of STAR alignment between all the samples. Outputted in aligned/LOGS folder. See ?STAR.multiQC

`keep.contaminants` logical, default FALSE. Create and keep contaminant aligning bam files, default is to only keep unaliged fastq reads, which will be further processed in "ge" genome alignment step. Useful if you want to do further processing on contaminants, like specific coverage of specific tRNAs etc.

`keep.unaligned.genome` logical, default FALSE. Create and keep reads that did not align at the genome alignment step, default is to only keep the aliged bam file. Useful if you want to do further processing on plasmids/custom sequences.

`script.folder` location of STAR index script, default internal ORFik file. You can change it and give your own if you need special alignments.

`script.single` location of STAR single file alignment script, default internal ORFik file. You can change it and give your own if you need special alignments.

## Details

Can only run on unix systems (Linux, Mac and WSL (Windows Subsystem Linux)), and requires a minimum of 30GB memory on genomes like human, rat, zebrafish etc.
If for some reason the internal STAR alignment bash script will not work for you, like if you want more customization of the STAR/fastp arguments. You can copy the internal alignment script, edit it and give that as the script used for this function.
The trimmer used is fastp (the fastest I could find), also works on (Linux, Mac and WSL (Windows Subsystem Linux)). If you want to use your own trimmer set file1/file2 to the location of the

trimmed files from your program.
A note on trimming from creator of STAR about trimming: "adapter trimming it definitely needed for short RNA sequencing. For long RNA-seq, I would agree with Devon that in most cases adapter trimming is not advantageous, since, by default, STAR performs local (not end-to-end) alignment, i.e. it auto-trims." So trimming can be skipped for longer reads.

## Value

output.dir, can be used as as input in ORFik::create.experiment

## See Also

Other STAR: STAR.align.single(), STAR.allsteps.multiQC(), STAR.index(), STAR.install(), STAR.multiQC(), STAR.remove.crashed.genome(), getGenomeAndAnnotation(), install.fastp()

## Examples

```
# First specify directories wanted (temp directory here)
config_file <- tempfile()
#config.save(config_file, base.dir = tempdir())
#config <- ORFik::config(config_file)

## Yeast RNA-seq samples (small genome)
#project <- ORFik::config.exper("chalmers_2012", "Saccharomyces_cerevisiae", "RNA-seq", config)
#annotation.dir <- project["ref"]
#fastq.input.dir <- project["fastq RNA-seq"]
#bam.output.dir <- project["bam RNA-seq"]

## Download some SRA data and metadata (subset to 50k reads)
# info <- download.SRA.metadata("SRP012047", outdir = conf["fastq RNA-seq"])
# info <- info[1:2,] # Subset to 2 first libraries
# download.SRA(info, fastq.input.dir, rename = FALSE, subset = 50000)

## No contaminant depletion:
# annotation <- getGenomeAndAnnotation("Saccharomyces cerevisiae", annotation.dir)
# index <- STAR.index(annotation)
# STAR.align.folder(fastq.input.dir, bam.output.dir,
#                   index, paired.end = FALSE) # Trim, then align to genome

## Human Ribo-seq sample (NB! very large genome and libraries!)
## Requires >= 32 GB memory
#project <- ORFik::config.exper("subtelny_2014", "Homo_sapiens", "Ribo-seq", config)
#annotation.dir <- project["ref"]
#fastq.input.dir <- project["fastq Ribo-seq"]
#bam.output.dir <- project["bam Ribo-seq"]

## Download some SRA data and metadata (full libraries)
# info <- download.SRA.metadata("DRR041459", fastq.input.dir)
# download.SRA(info, fastq.input.dir, rename = FALSE)
## Now align 2 different ways, without and with contaminant depletion

## No contaminant depletion:
```

```
# annotation <- getGenomeAndAnnotation("Homo sapiens", annotation.dir)
# index <- STAR.index(annotation)
# STAR.align.folder(fastq.input.dir, bam.output.dir,
#                   index, paired.end = FALSE)

## All contaminants merged:
# annotation <- getGenomeAndAnnotation(
#    organism = "Homo_sapiens",
#    phix = TRUE, ncRNA = TRUE, tRNA = TRUE, rRNA = TRUE,
#    output.dir = annotation.dir
#    )
# index <- STAR.index(annotation)
# STAR.align.folder(fastq.input.dir, bam.output.dir,
#                   index, paired.end = FALSE,
#                   steps = "tr-ge")
```

---

`STAR.align.single` *Align single or paired end pair with STAR*

---

## Description

Given a single NGS fastq/fasta library, or a paired setup of 2 mated libraries. Run either combination of fastq trimming, contamination removal and genome alignment. Works for (Linux, Mac and WSL (Windows Subsystem Linux))

## Usage

```
STAR.align.single(
  file1,
  file2 = NULL,
  output.dir,
  index.dir,
  star.path = STAR.install(),
  fastp = install.fastp(),
  steps = "tr-ge",
  adapter.sequence = "auto",
  quality.filtering = FALSE,
  min.length = 20,
  mismatches = 3,
  trim.front = 0,
  max.multimap = 10,
  alignment.type = "Local",
  allow.introns = TRUE,
  max.cpus = min(90, BiocParallel::bpparam()$workers),
  wait = TRUE,
  resume = NULL,
  keep.contaminants = FALSE,
  keep.unaligned.genome = FALSE,
```

```
  keep.index.in.memory = FALSE,
 script.single = system.file("STAR_Aligner", "RNA_Align_pipeline.sh", package = "ORFik")
)
```

## Arguments

| | |
|---|---|
| file1 | library file, if paired must be R1 file. Allowed formats are: (.fasta, .fastq, .fq, or.fa) with or without compression of .gz. This filename usually contains a suffix of .1 |
| file2 | default NULL, set if paired end to R2 file. Allowed formats are: (.fasta, .fastq, .fq, or.fa) with or without compression of .gz. This filename usually contains a suffix of .2 |
| output.dir | directory to save indices, default: paste0(dirname(arguments[1]), "/STAR_index/"), where arguments is the arguments input for this function. |
| index.dir | path to STAR index folder. Path returned from ORFik function STAR.index, when you created the index folders. |
| star.path | path to STAR, default: STAR.install(), if you don't have STAR installed at default location, it will install it there, set path to a runnable star if you already have it. |
| fastp | path to fastp trimmer, default: install.fastp(), if you have it somewhere else already installed, give the path. Only works for unix (linux or Mac OS), if not on unix, use your favorite trimmer and give the output files from that trimmer as input.dir here. |
| steps | a character, default: "tr-ge", trimming then genome alignment<br>steps of depletion and alignment wanted: The posible candidates you can use are:<br>• tr : trim reads<br>• co : contamination merged depletion<br>• ph : phix depletion<br>• rR : rrna depletion<br>• nc : ncrna depletion<br>• tR : trna depletion (Mature tRNA, so no intron checks done)<br>• ge : genome alignment<br>• all: run steps: "tr-co-ge" or "tr-ph-rR-nc-tR-ge", depending on if you have merged contaminants or not<br>If not "all", a subset of these ("tr-co-ph-rR-nc-tR-ge")<br>If co (merged contaminants) is used, non of the specific contaminants can be specified, since they should be a subset of co.<br>The step where you align to the genome is usually always included, unless you are doing pure contaminant analysis or only trimming. For Ribo-seq and TCP(RCP-seq) you should do rR (ribosomal RNA depletion), so when you made the STAR index you need the rRNA step, either use rRNA from .gtf or manual download. (usually just download a Silva rRNA database for SSU&LSU at: https://www.arb-silva.de/) for your species. |

`adapter.sequence`
: character, default: "auto". Auto detect adapter using fastp adapter auto detection, checking first 1.5M reads. (Auto detection of adapter will not work 100% of the time (if the library is of low quality), then you must rerun this function with specified adapter from fastp adapter analysis. , using FASTQC or other adapter detection tools, else alignment will most likely fail!). If already trimmed or trimming not wanted: adapter.sequence = "disable" .You can manually assign adapter like: "ATCTCGTATGCCGTCTTCTGCTTG" or "AAAAAAAAAAAAA". You can also specify one of the three presets:

  - illumina (TrueSeq ~75/100 bp sequencing): AGATCGGAAGAGC
  - small_RNA (standard for ~50 bp sequencing): TGGAATTCTCGG
  - nextera: CTGTCTCTTATA

  Paired end auto detection uses overlap sequence of pairs, to use the slower more secure paired end adapter detection, specify as: "autoPE".

`quality.filtering`
: logical, default FALSE. Not needed for modern library prep of RNA-seq, Ribo-seq etc (usually < ~ 0.5 If you are aligning bad quality data, set this to TRUE. These filters will then be applied (default of fastp), filter if:

  - Number of N bases in read: > 5
  - Read quality: > 40% of bases in the read are <Q15

`min.length`
: 20, minimum length of aligned read without mismatches to pass filter. Anything under 20 is dangerous, as chance of random hits will become high!

`mismatches`
: 3, max non matched bases. Excludes soft-clipping, this only filters reads that have defined mismatches in STAR. Only applies for genome alignment step.

`trim.front`
: 0, default trim 0 bases 5'. For Ribo-seq use default 0. Ignored if tr (trim) is not one of the arguments in "steps"

`max.multimap`
: numeric, default 10. If a read maps to more locations than specified, will skip the read. Set to 1 to only get unique mapping reads. Only applies for genome alignment step. The depletions are allowing for multimapping.

`alignment.type`
: default: "Local": standard local alignment with soft-clipping allowed, "EndToEnd" (global): force end-to-end read alignment, does not soft-clip.

`allow.introns`
: logical, default TRUE. Allow large gaps of N in reads during genome alignment, if FALSE: sets –alignIntronMax to 1 (no introns). NOTE: You will still get some spliced reads if you assigned a gtf at the index step.

`max.cpus`
: integer, default: `min(90, BiocParallel:::bpparam()$workers)`, number of threads to use. Default is minimum of 90 and maximum cores - 2. So if you have 8 cores it will use 6. Note: FASTP will use maximum 16 threads as from testing I see performance actually degrades using anything higher. From testing I also see STAR gets no performance gain after ~50 threads. I do suspect this will change when hard drives gets better in the future.

`wait`
: a logical (not `NA`) indicating whether the R interpreter should wait for the command to finish, or run it asynchronously. This will be ignored (and the interpreter will always wait) if `intern = TRUE`. When running the command asynchronously, no output will be displayed on the `Rgui` console in Windows (it will be dropped, instead).

`resume` default: NULL, continue from step, lets say steps are "tr-ph-ge": (trim, phix depletion, genome alignment) and resume is "ge", you will then use the assumed already trimmed and phix depleted data and start at genome alignment, useful if something crashed. Like if you specified wrong STAR version, but the trimming step was completed. Resume mode can only run 1 step at the time.

`keep.contaminants`
logical, default FALSE. Create and keep contaminant aligning bam files, default is to only keep unaliged fastq reads, which will be further processed in "ge" genome alignment step. Useful if you want to do further processing on contaminants, like specific coverage of specific tRNAs etc.

`keep.unaligned.genome`
logical, default FALSE. Create and keep reads that did not align at the genome alignment step, default is to only keep the aliged bam file. Useful if you want to do further processing on plasmids/custom sequences.

`keep.index.in.memory`
logical or character, default FALSE (i.e. LoadAndRemove). If TRUE, will keep index in memory, useful if you need to loop over single calls, instead of using STAR.align.folder (remember last run should use FALSE, to remove index). Alternative useful for MAC machines especially is "noShared", for machines that do not support shared memory index, usually gives error: "abort trap 6".

`script.single` location of STAR single file alignment script, default internal ORFik file. You can change it and give your own if you need special alignments.

## Details

Can only run on unix systems (Linux, Mac and WSL (Windows Subsystem Linux)), and requires a minimum of 30GB memory on genomes like human, rat, zebrafish etc.
If for some reason the internal STAR alignment bash script will not work for you, like if you want more customization of the STAR/fastp arguments. You can copy the internal alignment script, edit it and give that as the script used for this function.
The trimmer used is fastp (the fastest I could find), also works on (Linux, Mac and WSL (Windows Subsystem Linux)). If you want to use your own trimmer set file1/file2 to the location of the trimmed files from your program.
A note on trimming from creator of STAR about trimming: "adapter trimming it definitely needed for short RNA sequencing. For long RNA-seq, I would agree with Devon that in most cases adapter trimming is not advantageous, since, by default, STAR performs local (not end-to-end) alignment, i.e. it auto-trims." So trimming can be skipped for longer reads.

## Value

output.dir, can be used as as input in ORFik::create.experiment

## See Also

Other STAR: `STAR.align.folder()`, `STAR.allsteps.multiQC()`, `STAR.index()`, `STAR.install()`, `STAR.multiQC()`, `STAR.remove.crashed.genome()`, `getGenomeAndAnnotation()`, `install.fastp()`

## Examples

```
## Specify output libraries (using temp config)
config_file <- tempfile()
#config.save(config_file, base.dir = tempdir())
#config <- ORFik::config(config_file)
#project <- ORFik::config.exper("yeast_1", "Saccharomyces_cerevisiae", "RNA-seq", config)
# Get genome of yeast (quite small)
# arguments <- getGenomeAndAnnotation("Saccharomyces cerevisiae", project["ref"])
# index <- STAR.index(arguments)

## Make fake reads
#genome <- readDNAStringSet(arguments["genome"])
#which_chromosomes <- sample(seq_along(genome), 1000, TRUE, prob = width(genome))
#nt50_windows <- lapply(which_chromosomes, function(x)
# {window <- sample(width(genome[x]) - 51, 1); genome[[x]][seq(window, window+49)]})
#nt50_windows <- DNAStringSet(nt50_windows)
#names(nt50_windows) <- paste0("read_", seq_along(nt50_windows))
#dir.create(project["fastq RNA-seq"], recursive = TRUE)
#fake_fasta <- file.path(project["fastq RNA-seq"], "fake-RNA-seq.fasta")
#writeXStringSet(nt50_windows, fake_fasta, format = "fasta")
## Align the fake reads and import bam
# STAR.align.single(fake_fasta, NULL, project["bam RNA-seq"], index, steps = "ge")
#bam_file <- list.files(file.path(project["bam RNA-seq"], "aligned"),
#  pattern = "\.bam$", full.names = TRUE)
#fimport(bam_file)
```

---

`STAR.allsteps.multiQC` *Create STAR multiQC plot and table*

---

## Description

Takes a folder with multiple Log.final.out files from STAR, and create a multiQC report. This is automatically run with STAR.align.folder function.

## Usage

```
STAR.allsteps.multiQC(folder, steps = "auto", plot.ext = ".pdf")
```

## Arguments

| | |
|---|---|
| `folder` | path to main output folder of STAR run. The folder that contains /aligned/, "/trim/, "contaminants_depletion" etc. To find the LOGS folders in, to use for summarized statistics. |
| `steps` | a character, default "auto". Find which steps you did. If manual, a combination of "tr-co-ge". See STAR alignment functions for description. |
| `plot.ext` | character, default ".pdf". Which format to save QC plot. Alternative: ".png". |

## Value

data.table of main statistics, plots and data saved to disc. Named: "/00_STAR_LOG_plot.pdf" and "/00_STAR_LOG_table.csv"

## See Also

Other STAR: `STAR.align.folder()`, `STAR.align.single()`, `STAR.index()`, `STAR.install()`, `STAR.multiQC()`, `STAR.remove.crashed.genome()`, `getGenomeAndAnnotation()`, `install.fastp()`

---

`STAR.index` *Create STAR genome index*

---

## Description

Used as reference when aligning data
Get genome and gtf by running getGenomeAndFasta()

## Usage

```
STAR.index(
  arguments,
  output.dir = paste0(dirname(arguments[1]), "/STAR_index/"),
  star.path = STAR.install(),
  max.cpus = min(90, BiocParallel::bpparam()$workers),
  max.ram = 30,
  SAsparse = 1,
  tmpDirStar = "-",
  wait = TRUE,
  remake = FALSE,
  script = system.file("STAR_Aligner", "STAR_MAKE_INDEX.sh", package = "ORFik"),
  notify_load_existing = TRUE
)
```

## Arguments

| | |
|---|---|
| `arguments` | a named character vector containing paths wanted to use for index creation. They must be named correctly: names must be a subset of: c("gtf", "genome", "contaminants", "phix", "rRNA", "tRNA","ncRNA") |
| `output.dir` | directory to save indices, default: paste0(dirname(arguments[1]), "/STAR_index/"), where arguments is the arguments input for this function. |
| `star.path` | path to STAR, default: STAR.install(), if you don't have STAR installed at default location, it will install it there, set path to a runnable star if you already have it. |

max.cpus — integer, default: min(90, BiocParallel:::bpparam()$workers), number of threads to use. Default is minimum of 90 and maximum cores - 2. So if you have 8 cores it will use 6. Note: FASTP will use maximum 16 threads as from testing I see performance actually degrades using anything higher. From testing I also see STAR gets no performance gain after ~50 threads. I do suspect this will change when hard drives gets better in the future.

max.ram — integer, default 30, in Giga Bytes (GB). Maximum amount of RAM allowed for STAR limitGenomeGenerateRAM argument. RULE: idealy 10x genome size, but do not set too close to machine limit. Default fits well for human genome size (3 GB * 10 = 30 GB)

SAsparse — int > 0, default 1. If you do not have at least 64GB RAM, you might need to set this to 2. suffux array sparsity, i.e. distance between indices: use bigger numbers to decrease needed RAM at the cost of mapping speed reduction. Only applies to genome, not conaminants.

tmpDirStar — character, default "-". STAR automatic temp folder creation, deleted when done. The directory can not exists, as a safety STAR must make it!. If you are on a NFS file share drive, and you have a non NFS tmp dir, set this to tempfile() or the manually specified folder to get a considerable speedup!

wait — a logical (not NA) indicating whether the R interpreter should wait for the command to finish, or run it asynchronously. This will be ignored (and the interpreter will always wait) if intern = TRUE. When running the command asynchronously, no output will be displayed on the Rgui console in Windows (it will be dropped, instead).

remake — logical, default: FALSE, if TRUE remake everything specified

script — location of STAR index script, default internal ORFik file. You can change it and give your own if you need special alignments.

notify_load_existing — logical, default TRUE. If annotation exists (defined as: locally (a file called outputs.rds) exists in outputdir), print a small message notifying the user it is not redownloading. Set to FALSE, if this is not wanted

## Details

Can only run on unix systems (Linux and Mac), and requires minimum 30GB memory on genomes like human, rat, zebrafish etc.
If for some reason the internal STAR index bash script will not work for you, like if you have a very small genome. You can copy the internal index script, edit it and give that as the Index script used for this function. It is recommended to run through the RStudio local job tab, to give full info about the run. The system console will not stall, as can happen in happen in normal RStudio console.

## Value

output.dir, can be used as as input for STAR.align..

## See Also

Other STAR: STAR.align.folder(), STAR.align.single(), STAR.allsteps.multiQC(), STAR.install(), STAR.multiQC(), STAR.remove.crashed.genome(), getGenomeAndAnnotation(), install.fastp()

## Examples

```
## Manual way, specify all paths yourself.
#arguments <- c(path.GTF, path.genome, path.phix, path.rrna, path.trna, path.ncrna)
#names(arguments) <- c("gtf", "genome", "phix", "rRNA", "tRNA","ncRNA")
#STAR.index(arguments, "output.dir")

## Or use ORFik way:
output.dir <- "/Bio_data/references/Human"
# arguments <- getGenomeAndAnnotation("Homo sapiens", output.dir)
# STAR.index(arguments, output.dir)
```

---

# STAR.install *Download and prepare STAR*

---

## Description

Will not run "make", only use precompiled STAR file.
Can only run on unix systems (Linux and Mac), and requires minimum 30GB memory on genomes like human, rat, zebrafish etc.

## Usage

```
STAR.install(folder = "~/bin", version = "2.7.4a")
```

## Arguments

| | |
|---|---|
| folder | path to folder for download, fille will be named "STAR-version", where version is version wanted. |
| version | default "2.7.4a" |

## Details

ORFik for now only uses precompiled STAR binaries, so if you already have a STAR version it is adviced to redownload the same version, since STAR genome indices usually does not work between STAR versions.

## Value

path to runnable STAR

## References

https://www.ncbi.nlm.nih.gov/pubmed/23104886

## See Also

Other STAR: STAR.align.folder(), STAR.align.single(), STAR.allsteps.multiQC(), STAR.index(), STAR.multiQC(), STAR.remove.crashed.genome(), getGenomeAndAnnotation(), install.fastp()

## Examples

```
## Default folder install:
#STAR.install()
## Manual set folder:
folder <- "/I/WANT/IT/HERE"
#STAR.install(folder, version = "2.7.4a")
```

---

`STAR.multiQC` *Create STAR multiQC plot and table*

---

## Description

Takes a folder with multiple Log.final.out files from STAR, and create a multiQC report

## Usage

```
STAR.multiQC(folder, type = "aligned", plot.ext = ".pdf")
```

## Arguments

| | |
|---|---|
| `folder` | path to LOGS folder of ORFik STAR runs. Can also be the path to the aligned/ (parent directory of LOGS), then it will move into LOG from there. Only if no files with pattern Log.final.out are found in parent directory. If no LOGS folder is found it can check for a folder /aligned/LOGS/ so to go 2 folders down. |
| `type` | a character path, default "aligned". Which subfolder to check for. If you want log files for contamination do type = "contaminants_depletion" |
| `plot.ext` | character, default ".pdf". Which format to save QC plot. Alternative: ".png". |

## Value

a data.table with all information from STAR runs, plot and data saved to disc. Named: "/00_STAR_LOG_plot.pdf" and "/00_STAR_LOG_table.csv"

## See Also

Other STAR: `STAR.align.folder()`, `STAR.align.single()`, `STAR.allsteps.multiQC()`, `STAR.index()`, `STAR.install()`, `STAR.remove.crashed.genome()`, `getGenomeAndAnnotation()`, `install.fastp()`

## STAR.remove.crashed.genome *Remove crashed STAR genome*

### Description

This happens if you abort STAR run early, and it halts at: ..... loading genome

### Usage

```
STAR.remove.crashed.genome(index.path, star.path = STAR.install())
```

### Arguments

| | |
|---|---|
| index.path | path to index folder of genome |
| star.path | path to STAR, default: STAR.install(), if you don't have STAR installed at default location, it will install it there, set path to a runnable star if you already have it. |

### Value

return value from system call, 0 if all good.

### See Also

Other STAR: STAR.align.folder(), STAR.align.single(), STAR.allsteps.multiQC(), STAR.index(), STAR.install(), STAR.multiQC(), getGenomeAndAnnotation(), install.fastp()

### Examples

```
index.path = "/home/data/human_GRCh38/STAR_INDEX/genomeDir/"
# STAR.remove.crashed.genome(index.path = index.path)
## If you have the index argument from STAR.index function:
# index.path <- STAR.index()
# STAR.remove.crashed.genome(file.path(index.path, "genomeDir"))
# STAR.remove.crashed.genome(file.path(index.path, "contaminants_genomeDir"))
```

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

### Description

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

### Usage

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

### Arguments

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

### Value

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

### See Also

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

### Examples

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

---

## startDefinition *Returns start codon definitions*

---

### Description

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

### Usage

```
startDefinition(transl_table)
```

### Arguments

| | |
|---|---|
| transl_table | numeric. NCBI genetic code number for translation. |

### Value

A string of START sites separatd with "|".

## See Also

Other findORFs: `findMapORFs()`, `findORFs()`, `findORFsFasta()`, `findUORFs()`, `stopDefinition()`

## Examples

```
startDefinition
startDefinition(1)
```

---

`startRegion` *Start region as GRangesList*

---

## Description

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

## Usage

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

## Arguments

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

## Details

If tx is null, then upstream will be forced to 0 and downstream to a maximum of grl width (3' UTR end for mRNAs). Since there is no reference for splicing.

## Value

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

## See Also

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

## Examples

```
## ORF start region
orf <- GRangesList(tx1 = GRanges("1", 200:300, "+"))
tx <- GRangesList(tx1 = GRanges("1",
                   IRanges(c(100, 200), c(195, 400)), "+"))
startRegion(orf, tx, upstream = 6, downstream = 6)
## 2nd codon of ORF
startRegion(orf, tx, upstream = -3, downstream = 6)
```

---

`startRegionCoverage` *Start region coverage*

---

## Description

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

## Usage

```
startRegionCoverage(
  grl,
  RFP,
  tx = NULL,
  is.sorted = TRUE,
  upstream = 2L,
  downstream = 2L,
  weight = 1L
)
```

## Arguments

| | |
|---|---|
| `grl` | a `GRangesList` object with usually either leaders, cds', 3' utrs or ORFs |
| `RFP` | ribo seq reads as GAlignments, GRanges or GRangesList object |
| `tx` | default NULL, a GRangesList of transcripts or (container region), names of tx must contain all grl names. The names of grl can also be the ORFik orf names. that is "txName_id" |
| `is.sorted` | logical (TRUE), is grl sorted. |
| `upstream` | an integer (2), relative region to get upstream from. |
| `downstream` | an integer (2), relative region to get downstream from |
| `weight` | a vector (default: 1L, if 1L it is identical to countOverlaps()), if single number (!= 1), it applies for all, if more than one must be equal size of 'reads'. else it must be the string name of a defined meta column in subject "reads", that gives number of times a read was found. GRanges("chr1", 1, "+", score = 5), would mean "score" column tells that this alignment region was found 5 times. |

## Details

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

## Value

a numeric vector of counts

## See Also

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

## Examples

```
ORF <- GRanges(seqnames = "1",
               ranges = IRanges(21, 40),
               strand = "+")
names(ORF) <- c("tx1")
grl <- GRangesList(tx1 = ORF)
tx <- extendLeaders(grl, 20)
# 1 width p-shifted reads
reads <- GRanges("1", IRanges(c(21, 23, 50, 50, 50, 53, 53, 56, 59),
                              width = 1), "+")
score(reads) <- 28 # original width
startRegionCoverage(grl, reads, tx)
```

---

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

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| grl | a GRangesList of ranges to find regions in. |
| tx | a GRangesList of transcripts or (container region), names of tx must contain all gr names. The names of gr can also be the ORFik orf names. that is "tx-Name_id". |

| | |
|---|---|
| faFile | FaFile, BSgenome, fasta/index file path or an ORFik experiment. This file is usually used to find the transcript sequences from some GRangesList. |
| upstream | an integer, default (0), relative region to get upstream from. |
| downstream | an integer, default (0), relative region to get downstream from |

## Value

a character vector of start regions

---

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

---

## Description

In ATGTTTTGG, get the position of the A.

## Usage

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

## Arguments

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

## Value

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

## See Also

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

## Examples

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

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

## Description

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

## Usage

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

## Arguments

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

## Value

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

## See Also

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

## Examples

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

## stopDefinition *Returns stop codon definitions*

### Description

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

### Usage

```
stopDefinition(transl_table)
```

### Arguments

| | |
|---|---|
| transl_table | numeric. NCBI genetic code number for translation. |

### Value

A string of STOP sites separatd with "|".

### See Also

Other findORFs: `findMapORFs()`, `findORFs()`, `findORFsFasta()`, `findUORFs()`, `startDefinition()`

### Examples

```
stopDefinition
stopDefinition(1)
```

## stopRegion *Stop region as GRangesList*

### Description

Get the stop region of each ORF / region. If you want the stop codon only, set downstream = 0 or just use `stopCodons`. Standard is 2 upstream and 2 downstream, a width 5 window centered at stop site.

### Usage

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

### Arguments

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

### Details

If tx is null, then downstream will be forced to 0 and upstream to a minimum of -grl width (to the TSS). . Since there is no reference for splicing.

### Value

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

### See Also

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

### Examples

```
## ORF stop region
orf <- GRangesList(tx1 = GRanges("1", 200:300, "+"))
tx <- GRangesList(tx1 = GRanges("1",
                  IRanges(c(100, 305), c(300, 400)), "+"))
stopRegion(orf, tx, upstream = 6, downstream = 6)
## 2nd last codon of ORF
stopRegion(orf, tx, upstream = 6, downstream = -3)
```

---

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

---

### Description

In ATGTTTTGC, get the position of the C.

### Usage

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

### Arguments

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

### Value

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

### See Also

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

### Examples

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

---

## strandBool *Get logical list of strands*

---

### Description

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

### Usage

```
strandBool(grl)
```

### Arguments

| | |
|---|---|
| grl | a GRangesList or GRanges object |

### Value

a logical vector

## Examples

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

---

`strandMode,covRle-method`

*strandMode covRle*

---

## Description

strandMode covRle

## Usage

```
## S4 method for signature 'covRle'
strandMode(x)
```

## Arguments

| | |
|---|---|
| x | a covRle object |

## Value

integer vector with names

---

`strandMode,covRleList-method`

*strandMode covRle*

---

## Description

strandMode covRle

## Usage

```
## S4 method for signature 'covRleList'
strandMode(x)
```

## Arguments

| | |
|---|---|
| x | a covRle object |

## Value

integer vector with names

## 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, default: (TRUE) |

### 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)
```

## subsetToFrame *Subset GRanges to get desired frame.*

### Description

Usually used for ORFs to get specific frame (0-2): frame 0, frame 1, frame 2

### Usage

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

### Arguments

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

### Details

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

### Value

GRanges object reduced to only first frame

### Examples

```
subsetToFrame(GRanges("1", IRanges(1:10, width = 1), "+"), 2)
```

---

## symbols *Get ORFik experiment QC folder path*

---

### Description

Get ORFik experiment QC folder path

### Usage

```
symbols(x)
```

### Arguments

| | |
|---|---|
| x | an ORFik experiment |

### Value

a data.table with gene id, gene symbols and tx ids (3 columns)

---

## symbols,experiment-method *Get ORFik experiment QC folder path*

---

### Description

Get ORFik experiment QC folder path

### Usage

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

### Arguments

| | |
|---|---|
| x | an ORFik experiment |

## Value

a character path

---

`te.plot` *Translational efficiency plots*

---

## Description

Create 2 TE plots of:
- Within sample (TE log2 vs mRNA fpkm) ("default")
- Between all combinations of samples (x-axis: rna1fpkm - rna2fpkm, y-axis rfp1fpkm - rfp2fpkm)

## Usage

```
te.plot(
  df.rfp,
  df.rna,
  output.dir = QCfolder(df.rfp),
  type = c("default", "between"),
  filter.rfp = 1,
  filter.rna = 1,
  collapse = FALSE,
  plot.title = "",
  plot.ext = ".pdf",
  width = 6,
  height = "auto"
)
```

## Arguments

| | |
|---|---|
| `df.rfp` | a `experiment` of Ribo-seq or 80S from TCP-seq. |
| `df.rna` | a `experiment` of RNA-seq |
| `output.dir` | directory to save plots, plots will be named "TE_between.pdf" and "TE_within.pdf" |
| `type` | which plots to make, default: c("default", "between"). Both plots. |
| `filter.rfp` | numeric, default 1. minimum fpkm value to be included in plots |
| `filter.rna` | numeric, default 1. minimum fpkm value to be included in plots |
| `collapse` | a logical/character (default FALSE), if TRUE all samples within the group SAMPLE will be collapsed to one. If "all", all groups will be merged into 1 column called merged_all. Collapse is defined as rowSum(elements_per_group) / ncol(elements_per_group) |
| `plot.title` | title for plots, usually name of experiment etc |
| `plot.ext` | character, default: ".pdf". Alternatives: ".png" or ".jpg". |
| `width` | numeric, default 6 (in inches) |
| `height` | numeric or character, default "auto", which is: 3 + (ncol(RFP_CDS_FPKM)-2). Else a numeric value of height (in inches) |

## Details

Ribo-seq and RNA-seq must have equal nrows, with matching samples. Only exception is if RNA-seq is 1 single sample. Then it will use that for each of the Ribo-seq samples. Same stages, conditions etc, with a unique pairing 1 to 1. If not you can run collapse = "all". It will then merge all and do combined of all RNA-seq vs all Ribo-seq

## Value

a data.table with TE values, fpkm and log fpkm values, library samples melted into rows with split variable called "variable".

## Examples

```
##
# df.rfp <- read.experiment("zf_baz14_RFP")
# df.rna <- read.experiment("zf_baz14_RNA")
# te.plot(df.rfp, df.rna)
## Collapse replicates:
# te.plot(df.rfp, df.rna, collapse = TRUE)
```

---

# te.table *Create a TE table*

---

## Description

Creates a data.table with 6 columns, column names are:
variable, rfp_log2, rna_log2, rna_log10, TE_log2, id

## Usage

```
te.table(df.rfp, df.rna, filter.rfp = 1, filter.rna = 1, collapse = FALSE)
```

## Arguments

| | |
|---|---|
| df.rfp | a experiment of usually Ribo-seq or 80S from TCP-seq. (the numerator of the experiment, usually having a primary role) |
| df.rna | a experiment of usually RNA-seq. (the denominator of the experiment, usually having a normalizing function) |
| filter.rfp | numeric, default 1. What is the minimum fpkm value? |
| filter.rna | numeric, default 1. What is the minimum fpkm value? |
| collapse | a logical/character (default FALSE), if TRUE all samples within the group SAMPLE will be collapsed to one. If "all", all groups will be merged into 1 column called merged_all. Collapse is defined as rowSum(elements_per_group) / ncol(elements_per_group) |

## Value

a data.table with 6 columns

## See Also

Other DifferentialExpression: DEG.plot.static(), DEG_model(), DTEG.analysis(), DTEG.plot(), te_rna.plot()

## Examples

```
df <- ORFik.template.experiment()
df.rfp <- df[df$libtype == "RFP",]
df.rna <- df[df$libtype == "RNA",]
#te.table(df.rfp, df.rna)
```

---

# te_rna.plot *Translational efficiency plots*

---

## Description

Create TE plot of:
- Within sample (TE log2 vs mRNA fpkm)

## Usage

```
te_rna.plot(
  dt,
  output.dir = NULL,
  filter.rfp = 1,
  filter.rna = 1,
  plot.title = "",
  plot.ext = ".pdf",
  width = 6,
  height = "auto",
  dot.size = 0.4,
  xlim = c(filter.rna, filter.rna + 2.5)
)
```

## Arguments

| | |
|---|---|
| dt | a data.table with the results from te.table |
| output.dir | a character path, default NULL(no save), or a directory to save to a file will be called "TE_within.pdf" |
| filter.rfp | numeric, default 1. What is the minimum fpkm value? |
| filter.rna | numeric, default 1. What is the minimum fpkm value? |

| | |
|---|---|
| plot.title | title for plots, usually name of experiment etc |
| plot.ext | character, default: ".pdf". Alternatives: ".png" or ".jpg". |
| width | numeric, default 6 (in inches) |
| height | a numeric, width of plot in inches. Default "auto". |
| dot.size | numeric, default 0.4, size of point dots in plot. |
| xlim | numeric vector of length 2. X-axis limits. Default: `c(filter.rna, filter.rna + 2.5)` |

## Value

a ggplot object

## See Also

Other DifferentialExpression: `DEG.plot.static()`, `DEG_model()`, `DTEG.analysis()`, `DTEG.plot()`, `te.table()`

## Examples

```
df <- ORFik.template.experiment()
df.rfp <- df[df$libtype == "RFP",]
df.rna <- df[df$libtype == "RNA",]
#dt <- te.table(df.rfp, df.rna)
#te_rna.plot(dt, filter.rfp = 0, filter.rna = 5, dot.size = 1)
```

---

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

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| grl | a `GRangesList` object with names. |
| sort.on.return | logical (TRUE), should the groups be sorted before return (Negative ranges should be in decreasing order). Makes it a bit slower, but much safer for downstream analysis. |
| matchNaming | logical (TRUE), should groups keep unlisted names and meta data.(This make the list very big, for > 100K groups) |
| is.sorted | logical (TRUE), grl is presorted (negative coordinates are decreasing). Set to FALSE if they are not, else output will most likely be wrong! |

## Value

a GRangesList grouped by original group, tiled to 1. Groups with identical names will be merged.

## See Also

Other ExtendGenomicRanges: asTX(), coveragePerTiling(), extendLeaders(), extendTrailers(), 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)
```

---

# TOP.Motif.ecdf *TOP Motif ecdf plot*

---

## Description

Given sequences, DNA or RNA. And some score, scanning efficiency (SE), ribo-seq fpkm, TE etc.

## Usage

```
TOP.Motif.ecdf(
  seqs,
  rate,
  start = 1,
  stop = max(nchar(seqs)),
  xlim = c("q10", "q99"),
  type = "Scanning efficiency",
  legend.position.1st = c(0.75, 0.28),
  legend.position.motif = c(0.75, 0.28)
)
```

## Arguments

| | |
|---|---|
| seqs | the sequences (character vector, DNAStringSet), of 5' UTRs (leaders). See example below for input. |
| rate | a scoring vector (equal size to seqs) |

start position in seqs to start at (first is 1), default 1.

stop position in seqs to stop at (first is 1), default max(nchar(seqs)), that is the longest sequence length

xlim What interval of rate values you want to show type: numeric or quantile of length 2, 1. default c("q10","q99"). bigger than 10 percentile and less than 99 percentile. 2. Set to numeric values, like c(5, 1000), 3. Set to NULL if you want all values. Backend uses coord_cartesian.

type What type is the rate scoring ? default ("Scanning efficiency")

legend.position.1st
adjust left plot label position, default c(0.75, 0.28), ("none", "left", "right", "bottom", "top", or two-element numeric vector)

legend.position.motif
adjust right plot label position, default c(0.75, 0.28), ("none", "left", "right", "bottom", "top", or two-element numeric vector)

## Details

Top motif defined as a TSS of C and 4 T's or C's (pyrimidins) downstream of TSS C.

The right plot groups: C nucleotide, TOP motif (C, then 4 pyrimidines) and OTHER (all other TSS variants).

## Value

a ggplot gtable of the TOP motifs in 2 plots

## Examples

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

  # Should update by CAGE if not already done
  cageData <- system.file("extdata", "cage-seq-heart.bed.bgz",
                          package = "ORFik")
  leadersCage <- reassignTSSbyCage(leaders, cageData)
  # Get region to check
  seqs <- startRegionString(leadersCage, NULL,
        BSgenome.Hsapiens.UCSC.hg19::Hsapiens, 0, 4)
  # Some toy ribo-seq fpkm scores on cds
  set.seed(3)
  fpkm <- sample(1:115, length(leadersCage), replace = TRUE)
  # Standard arguments
  TOP.Motif.ecdf(seqs, fpkm, type = "ribo-seq FPKM",
                 legend.position.1st = "bottom",
                 legend.position.motif = "bottom")
  # with no zoom on x-axis:
```

```
  TOP.Motif.ecdf(seqs, fpkm, xlim = NULL,
                 legend.position.1st = "bottom",
                 legend.position.motif = "bottom")
}

## End(Not run)
```

---

topMotif *TOP Motif detection*

---

## Description

Per leader, detect if the leader has a TOP motif at TSS (5' end of leader) TOP motif defined as: (C, then 4 pyrimidines)

## Usage

```
topMotif(seqs, start = 1, stop = max(nchar(seqs)), return.sequence = TRUE)
```

## Arguments

| | |
|---|---|
| seqs | the sequences (character vector, DNAStringSet), of 5' UTRs (leaders) start region. seqs must be of minimum widths start - stop + 1 to be included. See example below for input. |
| start | position in seqs to start at (first is 1), default 1. |
| stop | position in seqs to stop at (first is 1), default max(nchar(seqs)), that is the longest sequence length |
| return.sequence | logical, default TRUE, return as data.table with sequence as columns in addition to TOP class. If FALSE, return character vector. |

## Value

default: return.sequence == FALSE, a character vector of either TOP, C or OTHER. C means leaders started on C, Other means not TOP and did not start on C. If return.sequence == TRUE, a data.table is returned with the base per position in the motif is included as additional columns (per position called seq1, seq2 etc) and a id column called X.gene_id (with names of seqs).

## Examples

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

```
  # Should update by CAGE if not already done
  cageData <- system.file("extdata", "cage-seq-heart.bed.bgz",
                          package = "ORFik")
  leadersCage <- reassignTSSbyCage(leaders, cageData)
  # Get region to check
  seqs <- startRegionString(leadersCage, NULL,
        BSgenome.Hsapiens.UCSC.hg19::Hsapiens, 0, 4)
  topMotif(seqs)
  }

## End(Not run)
```

---

`transcriptWindow` *Make 100 bases size meta window for all libraries in experiment*

---

## Description

Gives you binned meta coverage plots, either saved seperatly or all in one.

## Usage

```
transcriptWindow(
  leaders,
  cds,
  trailers,
  df,
  outdir = NULL,
  scores = c("sum", "transcriptNormalized"),
  allTogether = TRUE,
  colors = experiment.colors(df),
  title = "Coverage metaplot",
 windowSize = min(100, min(widthPerGroup(leaders, FALSE)), min(widthPerGroup(cds,
    FALSE)), min(widthPerGroup(trailers, FALSE))),
  returnPlot = is.null(outdir),
  dfr = NULL,
  idName = "",
  plot.ext = ".pdf",
  type = "ofst",
  is.sorted = FALSE,
  drop.zero.dt = TRUE,
  verbose = TRUE,
  force = TRUE,
  library.names = bamVarName(df),
  BPPARAM = bpparam()
)
```

## Arguments

| | |
|---|---|
| leaders | a GRangesList of leaders (5' UTRs) |
| cds | a GRangesList of coding sequences |
| trailers | a GRangesList of trailers (3' UTRs) |
| df | an ORFik experiment |
| outdir | directory to save to (default: NULL, no saving) |
| scores | scoring function (default: c("sum", "transcriptNormalized")), see ?coverageScorings for possible scores. |
| allTogether | plot all coverage plots in 1 output? (defualt: TRUE) |
| colors | Which colors to use, default auto color from function experiment.colors, new color per library type. Else assign colors yourself. |
| title | title of ggplot |
| windowSize | size of binned windows, default: 100 |
| returnPlot | return plot from function, default is.null(outdir), so TRUE if outdir is not defined. |
| dfr | an ORFik experiment of RNA-seq to normalize against. Will add RNA normalized to plot name if this is done. |
| idName | A character ID to add to saved name of plot, if you make several plots in the same folder, and same experiment, like splitting transcripts in two groups like targets / nontargets etc. (default: "") |
| plot.ext | character, default: ".pdf". Alternatives: ".png" or ".jpg". |
| type | a character(default: "default"), load files in experiment or some precomputed variant, like "ofst" or "pshifted". These are made with ORFik:::convertLibs(), shiftFootprintsByExperiment(), etc. Can also be custom user made folders inside the experiments bam folder. It acts in a recursive manner with priority: If you state "pshifted", but it does not exist, it checks "ofst". If no .ofst files, it uses "default", which always must exists.<br>Presets are (folder is relative to default lib folder, some types fall back to other formats if folder does not exist):<br>- "default": load the original files for experiment, usually bam.<br>- "ofst": loads ofst files from the ofst folder, relative to lib folder (falls back to default)<br>- "pshifted": loads ofst, wig or bigwig from pshifted folder (falls back to ofst, then default)<br>- "cov": Load covRle objects from cov_RLE folder (fail if not found)<br>- "covl": Load covRleList objects, from cov_RLE_List folder (fail if not found)<br>- "bed": Load bed files, from bed folder (falls back to default)<br>- Other formats must be loaded directly with fimport |
| is.sorted | logical (FALSE), is grl sorted. That is + strand groups in increasing ranges (1,2,3), and - strand groups in decreasing ranges (3,2,1) |
| drop.zero.dt | logical FALSE, if TRUE and as.data.table is TRUE, remove all 0 count positions. This greatly speeds up and most importantly, greatly reduces memory usage. Will not change any plots, unless 0 positions are used in some sense. (mean, median, zscore coverage will only scale differently) |

| | |
|---|---|
| verbose | logical, default TRUE, message about library output status. |
| force | logical, default TRUE If TRUE, reload library files even if matching named variables are found in environment used by experiment (see envExp) A simple way to make sure correct libraries are always loaded. FALSE is faster if data is loaded correctly already. |
| BPPARAM | how many cores/threads to use? default: bpparam() |

## Value

NULL, or ggplot object if returnPlot is TRUE

## See Also

Other experiment plots: transcriptWindow1(), transcriptWindowPer()

## Examples

```
df <- ORFik.template.experiment()[3,] # Only third library
loadRegions(df) # Load leader, cds and trailers as GRangesList
#transcriptWindow(leaders, cds, trailers, df, outdir = "directory/to/save")
```

---

# 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,
  librarySize = "full",
  weight.RFP = 1L,
  weight.RNA = 1L
)
```

## Arguments

| | |
|---|---|
| grl | a GRangesList object can be either transcripts, 5' utrs, cds', 3' utrs or ORFs as a special case (uORFs, potential new cds' etc). If regions are not spliced you can send a GRanges object. |
| RNA | RnaSeq reads as GAlignments, GRanges or GRangesList object |
| RFP | RiboSeq reads as GAlignments, 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, default: FALSE, if true return the fpkm values together with translational efficiency as a data.table |
| pseudoCount | an integer, by default is 0, set it to 1 if you want to avoid NA and inf values. |
| librarySize | either numeric value or character vector. Default ("full"), number of alignments in library (reads). If you just have a subset, you can give the value by librarySize = length(wholeLib), if you want lib size to be only number of reads overlapping grl, do: librarySize = "overlapping" sum(countOverlaps(reads, grl) > 0), if reads[1] has 3 hits in grl, and reads[2] has 2 hits, librarySize will be 2, not 5. You can also get the inverse overlap, if you want lib size to be total number of overlaps, do: librarySize = "DESeq" This is standard fpkm way of DESeq2::fpkm(robust = FALSE) sum(countOverlaps(grl, reads)) if grl[1] has 3 reads and grl[2] has 2 reads, librarySize is 5, not 2. |
| weight.RFP | a vector (default: 1L). Can also be character name of column in RFP. As in translationalEff(weight = "score") for: GRanges("chr1", 1, "+", score = 5), would mean score column tells that this alignment region was found 5 times. |
| weight.RNA | Same as weightRFP but for RNA weights. (default: 1L) |

## Value

a numeric vector of fpkm ratios, if with.fpkm is TRUE, return a data.table with te and fpkm values (total 3 columns then)

## References

doi: 10.1126/science.1168978

## See Also

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

---

# trimming.table *Create trimming table*

---

## Description

From fastp runs in ORFik alignment process

## Usage

```
trimming.table(trim_folder)
```

## Arguments

| | |
|---|---|
| trim_folder | folder of trimmed files, only reads fastp .json files |

## Value

a data.table with 6 columns, raw_library (names of library), raw_reads (numeric, number of raw reads), trim_reads (numeric, number of trimmed reads), raw_mean_length (numeric, raw mean read length), trim_mean_length (numeric, trim mean read length).

## Examples

```
# Location of fastp trimmed .json files
trimmed_folder <- "path/to/libraries/trim/"
#trimming.table(trimmed_folder)
```

txNames *Get transcript names from orf names*

## Description

Using the ORFik definition of orf name, which is: example ENSEMBL:
tx name: ENST0909090909090
orf id: _1 (the first of on that tx)
orf_name: ENST0909090909090_1
So therefor txNames("ENST0909090909090_1") = ENST0909090909090

## Usage

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

## Arguments

| | |
|---|---|
| grl | a GRangesList grouped by ORF , GRanges object or IRanges object. |
| ref | a reference GRangesList. The object you want grl to subset by names. Add to make sure naming is valid. |
| unique | a boolean, if true unique the names, used if several orfs map to same transcript and you only want the unique groups |

## Details

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

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

## Value

a character vector of transcript names, without _* naming

## See Also

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

## Examples

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

---

# txNamesToGeneNames *Convert transcript names to gene names*

---

## Description

Works for ensembl, UCSC and other standard annotations.

## Usage

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

## Arguments

| | |
|---|---|
| txNames | character vector, the transcript names to convert. Can also be a named object with tx names (like a GRangesList), will then extract names. |
| txdb | the transcript database to use or gtf/gff path to it. |

## Value

character vector of gene names

## Examples

```
df <- ORFik.template.experiment()
txdb <- loadTxdb(df)
loadRegions(txdb, "cds") # using tx names
txNamesToGeneNames(cds, txdb)
# Identical to:
loadRegions(txdb, "cds", by = "gene")
```

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

## Description

For each GRanges object, find the sequence of it from faFile or BSgenome.

## Usage

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

## Arguments

| | |
|---|---|
| grl | a GRangesList object |
| faFile | FaFile, BSgenome, fasta/index file path or an ORFik experiment. This file is usually used to find the transcript sequences from some GRangesList. |
| is.sorted | a speedup, if you know the grl ranges are sorted |
| keep.names | a logical, default (TRUE), if FALSE: return as character vector without names. |

## Details

A wrapper around extractTranscriptSeqs that works for DNAStringSet and ORFik experiment input. For debug of errors do: which(!(unique(seqnamesPerGroup(grl, FALSE)) This happens usually when the grl contains chromsomes that the fasta file does not have. A normal error is that mitocondrial chromosome is called MT vs chrM even though they have same seqlevelsStyle. The above line will give you which chromosome it is missing.

## Value

a DNAStringSet of the transcript sequences

## See Also

Other ExtendGenomicRanges: asTX(), coveragePerTiling(), extendLeaders(), extendTrailers(), reduceKeepAttr(), tile1(), windowPerGroup()

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

### Description

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

### Usage

```
uniqueGroups(grl)
```

### Arguments

| | |
|---|---|
| grl | a GRangesList |

### Value

a GRangesList of unique orfs

### See Also

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

### Examples

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

## uniqueOrder *Get unique ordering for GRangesList groups*

### Description

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

### Usage

```
uniqueOrder(grl)
```

### Arguments

| | |
|---|---|
| grl | a GRangesList |

### Value

an integer vector of indices of unique groups

### See Also

uniqueGroups

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

### Examples

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

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

---

## unlistGrl *Safe unlist*

---

### Description

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

### Usage

```
unlistGrl(grl)
```

### Arguments

| | |
|---|---|
| grl | a GRangesList |

### Value

a GRanges object

## Examples

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

---

`uORFSearchSpace` *Create search space to look for uORFs*

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| `fiveUTRs` | (GRangesList) The 5' leaders or full transcript sequences |
| `cage` | Either a filePath for the CageSeq file as .bed .bam or .wig, with possible compressions (".gzip", ".gz", ".bgz"), or already loaded CageSeq peak data as GRanges or GAlignment. NOTE: If it is a .bam file, it will add a score column by running: convertToOneBasedRanges(cage, method = "5prime", addScoreColumn = TRUE) The score column is then number of replicates of read, if score column is something else, like read length, set the score column to NULL first. |
| `extension` | The maximum number of basses upstream of the TSS to search for CageSeq peak. |
| `filterValue` | The minimum number of reads on cage position, for it to be counted as possible new tss. (represented in score column in CageSeq data) If you already filtered, set it to 0. |

restrictUpstreamToTx
: a logical (FALSE). If TRUE: restrict leaders to not extend closer than 5 bases from closest upstream leader, set this to TRUE.

removeUnused
: logical (FALSE), if False: (standard is to set them to original annotation), If TRUE: remove leaders that did not have any cage support.

cds
: (GRangesList) CDS of relative fiveUTRs, applicable only if you want to extend 5' leaders downstream of CDS's, to allow upstream ORFs that can overlap into CDS's.

## Value

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

## See Also

Other uorfs: addCdsOnLeaderEnds(), filterUORFs(), removeORFsWithSameStartAsCDS(), removeORFsWithSameStopA
removeORFsWithStartInsideCDS(), removeORFsWithinCDS()

## Examples

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

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

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

---

# 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, default: (TRUE) |

### Value

an integer vector (named/unnamed) of widths

### Examples

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

---

## windowCoveragePlot *Get meta coverage plot of reads*

---

### Description

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

### Usage

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

### Arguments

| | |
|---|---|
| coverage | a data.table, e.g. output of scaledWindowCoverage |
| output | character string (NULL), if set, saves the plot as pdf or png to path given. If no format is given, is save as pdf. |
| scoring | character vector, default "zscore", either of zscore, transcriptNormalized, sum, mean, median, .. or NULL. Set NULL if already scored. see ?coverageScorings for info and more alternatives. |

| | |
|---|---|
| colors | character vector colors to use in plot, will fix automaticly, using binary splits with colors c('skyblue4', 'orange'). |
| title | a character (metaplot) (what is the title of plot?) |
| type | a character (transcripts), what should legends say is the whole region? Transcripts, genes, non coding rnas etc. |
| scaleEqual | a logical (FALSE), should all fractions (rows), have same max value, for easy comparison of max values if needed. |
| setMinToZero | a logical (FALSE), should minimum y-value be 0 (TRUE). With FALSE minimum value is minimum score at any position. This parameter overrides scaleEqual. |

## Details

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

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

## Value

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

## See Also

Other coveragePlot: `coverageHeatMap()`, `pSitePlot()`, `savePlot()`

## Examples

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

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

# See vignette for a more practical example
```

windowPerGroup *Get window region of GRanges object*

## Description

Per GRanges input (gr) of single position inputs (center point), create a GRangesList window output of specified upstream, downstream region relative to some transcript "tx".
If downstream is 20, it means the window will start 20 downstream of gr start site (-20 in relative transcript coordinates.) If upstream is 20, it means the window will start 20 upstream of gr start site (+20 in relative transcript coordinates.) It will keep exon structure of tx, so if -20 is on next exon, it jumps to next exon.

## Usage

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

## Arguments

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

## Details

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

## Value

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

## See Also

Other ExtendGenomicRanges: `asTX()`, `coveragePerTiling()`, `extendLeaders()`, `extendTrailers()`, `reduceKeepAttr()`, `tile1()`, `txSeqsFromFa()`

## Examples

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

# With multiple extensions downstream
ORF <- rep(ORF, 2)
names(ORF)[2] <- "tx1_2"
windowPerGroup(ORF, tx, upstream = 0, downstream = c(2, 5))
# The last one gives 2nd and (1st and 2nd) codon as two groups
```

---

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

---

## Description

This is defined as: Fraction of reads per read length, per position in whole window (defined by upstream and downstream) If tx is not NULL, it gives a metaWindow, centered around startSite of grl from upstream and downstream. If tx is NULL, it will use only downstream , since it has no reference on how to find upstream region. The exception is when upstream is negative, that is, going into downstream region of the object.

## Usage

```
windowPerReadLength(
  grl,
  tx = NULL,
  reads,
  pShifted = TRUE,
 upstream = ifelse(!is.null(tx), ifelse(pShifted, 5, 20), min(ifelse(pShifted, 5, 20),
    0)),
  downstream = ifelse(pShifted, 20, 5),
  acceptedLengths = NULL,
  zeroPosition = upstream,
  scoring = "transcriptNormalized",
  weight = "score",
  drop.zero.dt = FALSE,
  append.zeroes = FALSE,
  windows = startRegion(grl, tx, TRUE, upstream, downstream)
)
```

## Arguments

| | |
|---|---|
| `grl` | a `GRangesList` object with usually either leaders, cds', 3' utrs or ORFs |

| | |
|---|---|
| tx | default NULL, a GRangesList of transcripts or (container region), names of tx must contain all grl names. The names of grl can also be the ORFik orf names. that is "txName_id" |
| reads | a GAlignments, GRanges, or precomputed coverage as covRleList (where names of covRle objects are readlengths) of RiboSeq, RnaSeq etc.<br>Weigths for scoring is default the 'score' column in 'reads'. Can also be random access paths to bigWig or fstwig file. Do not use random access for more than a few genes, then loading the entire files is usually better. |
| pShifted | a logical (TRUE), are Ribo-seq reads p-shifted to size 1 width reads? If upstream and downstream is set, this argument is irrelevant. So set to FALSE if this is not p-shifted Ribo-seq. |
| upstream | an integer (5), relative region to get upstream from. Default: `ifelse(!is.null(tx), ifelse(pShifted, 5, 20), min(ifelse(pShifted, 5, 20), 0))` |
| downstream | an integer (20), relative region to get downstream from. Default: `ifelse(pShifted, 20, 5)` |
| acceptedLengths | an integer vector (NULL), the read lengths accepted. Default NULL, means all lengths accepted. |
| zeroPosition | an integer DEFAULT (upstream), what is the center point? Like leaders and cds combination, then 0 is the TIS and -1 is last base in leader. NOTE!: if windows have different widths, this will be ignored. |
| scoring | a character (transcriptNormalized), which meta coverage scoring ? one of (zscore, transcriptNormalized, mean, median, sum, sumLength, fracPos), see ?coverageScorings for more info. Use to decide a scoring of hits per position for metacoverage etc. Set to NULL if you do not want meta coverage, but instead want per gene per position raw counts. |
| weight | (default: 'score'), if defined a character name of valid meta column in subject. GRanges("chr1", 1, "+", score = 5), would mean score column tells that this alignment region was found 5 times. ORFik ofst, bedoc and .bedo files contains a score column like this. As do CAGEr CAGE files and many other package formats. You can also assign a score column manually. |
| drop.zero.dt | logical FALSE, if TRUE and as.data.table is TRUE, remove all 0 count positions. This greatly speeds up and most importantly, greatly reduces memory usage. Will not change any plots, unless 0 positions are used in some sense. (mean, median, zscore coverage will only scale differently) |
| append.zeroes | logical, default FALSE. If TRUE and drop.zero.dt is TRUE and all windows have equal length, it will add back 0 values after transformation. Sometimes needed for correct plots, if TRUE, will call abort if not all windows are equal length! |
| windows | the GRangesList windows to actually check, default: `startRegion(grl, tx, TRUE, upstream, downstream)`. |

## Details

Careful when you create windows where not all transcripts are long enough, this function usually is used first with filterTranscripts to make sure they are of all of valid length!

## Value

a data.table with 4 columns: position (in window), score, fraction (read length). If score is NULL, will also return genes (index of grl). A note is that if no coverage is found, it returns an empty data.table.

## See Also

Other coverage: coverageScorings(), metaWindow(), regionPerReadLength(), scaledWindowPositions()

## Examples

```
cds <- GRangesList(tx1 = GRanges("1", 100:129, "+"))
tx <- GRangesList(tx1 = GRanges("1", 80:129, "+"))
reads <- GRanges("1", seq(79,129, 3), "+")
windowPerReadLength(cds, tx, reads, scoring = "sum")
windowPerReadLength(cds, tx, reads, scoring = "transcriptNormalized")
```

# Index