Package ‘coRdon’

September 22, 2024

Type Package

Title Codon Usage Analysis and Prediction of Gene Expressivity

Version 1.23.0

Description Tool for analysis of codon usage in various unannotated or KEGG/COG annotated DNA sequences. Calculates different measures of CU bias and CU-based predictors of gene expressivity, and performs gene set enrichment analysis for annotated sequences. Implements several methods for visualization of CU and enrichment analysis results.

License Artistic-2.0

LazyData TRUE

biocViews Software, Metagenomics, GeneExpression, GeneSetEnrichment, GenePrediction, Visualization, KEGG, Pathways, Genetics CellBiology, BiomedicalInformatics, ImmunoOncology

Depends R (>= 3.5)

Imports methods, stats, utils, Biostrings, Biobase, dplyr, stringr, purrr, ggplot2, data.table

Suggests BiocStyle, testthat, knitr, rmarkdown

RoxygenNote 6.1.1

Collate 'coRdon.R' 'genCode-class.R' 'codonTable-class.R' 'functions.R' 'codonUsage.R' 'codonUsage-expressivity.R' 'codonUsage-visualization.R' 'crossTab-class.R' 'data.R' 'enrichment-visualization.R' 'enrichment.R' 'readSet.R'

VignetteBuilder knitr

URL https://github.com/BioinfoHR/coRdon

BugReports https://github.com/BioinfoHR/coRdon/issues

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

git_branch devel

git_last_commit 225be74

git_last_commit_date 2024-04-30

Repository Bioconductor 3.20

Date/Publication 2024-09-22

Author Anamaria Elek [cre, aut],
Maja Kuzman [aut],
Kristian Vlahovicek [aut]

Maintainer Anamaria Elek <anamariaelek@gmail.com>

Contents

`Bplot` *Karlin B plot*

Description

Plot distances of each gene's CU frequency to specified gene (sub)sets (given by `x` and `y`).

Usage

```
Bplot(x, y, data, annotations = character(), ribosomal = FALSE,
  reference = list(), size = 1, alpha = 0.5)

## S4 method for signature 'character,character,matrix'
Bplot(x, y, data,
  annotations = character(), ribosomal = FALSE, reference = list(),
```

```
  size = 1, alpha = 0.5)

## S4 method for signature 'numeric,numeric,missing'
Bplot(x, y, data,
  annotations = character(), ribosomal = FALSE, reference = list(),
  size = 1, alpha = 0.5)
```

Arguments

x, y	Character, both must be in `colnames(data)`, or numeric vectors of CU statistic values for two subsets of genes. If numeric, the vectors must be of the same length.
data	A matrix with CU statistic values for subsets of genes in columns.
annotations	A character vector giving KO annotations for sequences for which the CU values were calculated, must be of length `nrow(data)`.
ribosomal	Logical, whether to indicate ribosomal genes in the plot. Default is `FALSE`, if set to `TRUE`, then `annotation` must be given.
reference	A named list of length 1, containing either a logical vector of `nrow(data)` of reference genes to be indicated on the plot, or a character vector (of any length) of the reference genes' anotations. If latter is the case, then `annotation` must be given.
size	Numeric, indicating points' size
alpha	Numeric, between 0 and 1, indicating points' transparency (default is 0.1).

Value

A `ggplot` object.

Examples

```
require(ggplot2)

# calculate MILC distance to the average CU of the example DNA sequences,
# and to the average CU of ribosomal genes among the example DNA sequences
milc <- MILC(LD94, self = TRUE, ribosomal = TRUE)

Bplot(x = "ribosomal", y = "self", data = milc,
      ribosomal = TRUE, annotations = getKO(LD94),
      size = 3) +
    labs(x = "MILC distance to ribosomal genes",
         y = "MILC distance to genes' average CU")
```

---

codonTable-class *An S4 class* codonTable

Description

Contains codon counts and optional annotation for a set DNA sequences.

Usage

```
codonTable(x)

## S4 method for signature 'DNAStringSet'
codonTable(x)

## S4 method for signature 'matrix'
codonTable(x)

## S4 method for signature 'data.frame'
codonTable(x)

codonCounts(object)

## S4 method for signature 'codonTable'
codonCounts(object)

getID(object)

## S4 method for signature 'codonTable'
getID(object)

getlen(object)

## S4 method for signature 'codonTable'
getlen(object)

getKO(object)

## S4 method for signature 'codonTable'
getKO(object)

setKO(object, ann)

## S4 method for signature 'codonTable'
setKO(object, ann)

getCOG(object)

## S4 method for signature 'codonTable'
getCOG(object)

setCOG(object, ann)

## S4 method for signature 'codonTable'
setCOG(object, ann)
```

Arguments

`x`	An object of `DNAStringSet`, `matrix` or `data.frame` class.
`object`	A `codonTable` object.
`ann`	A character vector of sequence annotations, must be of length equal to `length(object)`.

Value

A `codonTable`.

Methods (by generic)

- `codonTable`: Create new objects of class `codonTable`.
- `codonCounts`: Get codon counts from `codonTable` object.
- `getID`: Get IDs for `codonTable` class.
- `getlen`: Get lengths of sequences in `codonTable` object.
- `getKO`: Get KO annotations of sequences in `codonTable` object.
- `setKO`: Set KO annotations for `codonTable` object.
- `getCOG`: Get COG annotations of sequences in `codonTable` object.
- `setCOG`: Set COG annotations for `codonTable` object.

Slots

`ID` A character vector of sequence identifiers.

`counts` A matrix containing codon counts. Columns are codons, rows are sequences.

`len` A numeric vector,length equal to `nrow(counts)`, containing lengths of sequnces.

`KO` A character vector of KEGG annotations for sequences, length equal to `nrow(counts)`. If no annotation is available, this will be an empty vector.

`COG` A character vector of COG annotations for sequences, length equal to `nrow(counts)`. If no annotation is available, this will be an empty vector.

Examples

```
# create codonTable with codon counts for sequences in DNAStringSet
require(Biostrings)
dna <- DNAStringSet(c("ACGAAGTGTACTGTAATTTGCACAGTACTTAAATGT",
                      "ACGTCCGTACTGATCGATTCCGTGATT"))
cT <- codonTable(dna)
codonCounts(cT)
getlen(cT)
getKO(cT)
cT <- setKO(cT, c("K00001", "K00002"))
getKO(cT)

# convert matrix containing codon counts to codonTable
mat <- matrix(sample(1:10, 122, replace = TRUE), nrow = 2)
codonTable(mat) # produces informative warning
```

codonUsage *Calculate CU measures.*

Description

Calculate values of the codon usage (CU) measure for every sequence in the given codonTable object. The following methods are implemented: MILC, Measure Independent of Length and Composition Supek & Vlahovicek (2005), B, codon usage bias (B) Karlin et al. (2001), ENC, effective number of codons (ENC) Wright (1990). ENCprime, effective number of codons prime (ENC') Novembre (2002), MCB, maximum-likelihood codon bias (MCB) Urrutia and Hurst (2001), SCUO, synonymous codon usage eorderliness (SCUO) Wan et al. (2004).

Usage

```
MILC(cTobject, subsets = list(), self = TRUE, ribosomal = FALSE,
  id_or_name2 = "1", alt.init = TRUE, stop.rm = FALSE,
  filtering = "none", len.threshold = 80)

## S4 method for signature 'codonTable'
MILC(cTobject, subsets = list(), self = TRUE,
  ribosomal = FALSE, id_or_name2 = "1", alt.init = TRUE,
  stop.rm = FALSE, filtering = "none", len.threshold = 80)

B(cTobject, subsets = list(), self = TRUE, ribosomal = FALSE,
  id_or_name2 = "1", alt.init = TRUE, stop.rm = FALSE,
  filtering = "none", len.threshold = 80)

## S4 method for signature 'codonTable'
B(cTobject, subsets = list(), self = TRUE,
  ribosomal = FALSE, id_or_name2 = "1", alt.init = TRUE,
  stop.rm = FALSE, filtering = "none", len.threshold = 80)

MCB(cTobject, subsets = list(), self = TRUE, ribosomal = FALSE,
  id_or_name2 = "1", alt.init = TRUE, stop.rm = FALSE,
  filtering = "none", len.threshold = 80)

## S4 method for signature 'codonTable'
MCB(cTobject, subsets = list(), self = TRUE,
  ribosomal = FALSE, id_or_name2 = "1", alt.init = TRUE,
  stop.rm = FALSE, filtering = "none", len.threshold = 80)

ENCprime(cTobject, subsets = list(), self = TRUE, ribosomal = FALSE,
  id_or_name2 = "1", alt.init = TRUE, stop.rm = TRUE,
  filtering = "none", len.threshold = 80)

## S4 method for signature 'codonTable'
ENCprime(cTobject, subsets = list(),
  self = TRUE, ribosomal = FALSE, id_or_name2 = "1",
  alt.init = TRUE, stop.rm = TRUE, filtering = "none",
  len.threshold = 80)
```

```
ENC(cTobject, id_or_name2 = "1", alt.init = TRUE, stop.rm = TRUE,
  filtering = "none", len.threshold = 80)

## S4 method for signature 'codonTable'
ENC(cTobject, id_or_name2 = "1",
  alt.init = TRUE, stop.rm = TRUE, filtering = "none",
  len.threshold = 80)

SCUO(cTobject, id_or_name2 = "1", alt.init = TRUE, stop.rm = FALSE,
  filtering = "none", len.threshold = 80)

## S4 method for signature 'codonTable'
SCUO(cTobject, id_or_name2 = "1",
  alt.init = TRUE, stop.rm = FALSE, filtering = "none",
  len.threshold = 80)
```

Arguments

`cTobject`	A `codonTable` object.
`subsets`	A (named) list of logical vectors, the length of each equal to `getlen(cTobject)`, i.e. the number of sequences in the set, or character vectors (of any length) containing KEGG/eggNOG annotations, or codonTable objects (of any length). Not used for `ENC`, `SCUO` and `GCB` calculations.
`self`	Logical, if `TRUE` (default), CU statistic is also calculated against the average CU of the entire set of sequences. Not used for `ENC`, `SCUO` and `GCB` calculations.
`ribosomal`	Logical, if `TRUE`, CU statistic is also calculated against the average CU of the ribosomal genes in the sequence set. Not used for `ENC` and `SCUO` calculations. For GCB calculations, if `TRUE`, ribosomal genes are used as a seed, and if `FALSE` (default), `seed` has to be specified.
`id_or_name2`	A single string that uniquely identifies the genetic code to extract. Should be one of the values in the `id` or `name2` columns of `GENETIC_CODE_TABLE`.
`alt.init`	logical, whether to use alternative initiation codons. Default is `TRUE`.
`stop.rm`	Logical, whether to remove stop codons. Default is `FALSE`.
`filtering`	Character vector, one of `c("none", "soft", "hard")`. Specifies whether sequences shorther than some threshold value of length (in codons), `len.threshold`, should be excluded from calculations. If `"none"` (default), length of sequences is not checked, if `"soft"`, a warrning is printed if there are shorter sequences, and if `"hard"`, these sequences are excluded from calculation.
`len.threshold`	Optional numeric, specifying sequence length, in codons, used for filtering.

Value

A matrix or a numeric vector with CU measure values. For `MILC`, `B`, `ENCprime`, the matrix has a column with values for every specified subset (`subsets`, `self`, `ribosomal`). A numeric vector for `ENC` and `SCUO`.

Examples

```
# load example DNA sequences
exampledir <- system.file("extdata", package = "coRdon")
cT <- codonTable(readSet(exampledir))
```

```
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# In the examples below, MILC values are calculated for all sequences;
# B and ENCprime can be caluclated in the same way.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

# calculate MILC distance to the average CU of the example DNA sequences
milc <- MILC(cT)
head(milc)

# also calculate MILC distance to the average CU
# of ribosomal genes among the example DNA sequences
milc <- MILC(cT, ribosomal = TRUE)
head(milc)

# calculate MILC distance to the average CU
# of the first 20 example DNA sequences
# (i.e. the first half of the example DNA set)
milc <- MILC(cT, self = FALSE,
             subsets = list(half = c(rep(TRUE, 20), rep(FALSE, 20))))

# alternatively, you can specify codonTable as a subset
halfcT <- codonTable(codonCounts(cT)[1:20,])
milc2 <- MILC(cT, self = FALSE, subsets = list(half = halfcT))
all.equal(milc, milc2) # TRUE

# filtering
MILC(cT, filtering = "hard", len.threshold = 80) # MILC for 9 sequences
sum(getlen(cT) > 80) # 9 sequences are longer than 80 codons
milc1 <- MILC(cT, filtering = "none") # no filtering
milc2 <- MILC(cT, filtering = "soft") # warning
all.equal(milc1, milc2) # TRUE

# options for genetic code
milc <- MILC(cT, stop.rm = TRUE) # don't use stop codons in calculation
milc <- MILC(cT, alt.init = FALSE) # don't use alternative start codons
milc <- MILC(cT, id_or_name2 = "2") # use different genetic code, for help
                                    # see `?Biostrings::GENETIC_CODE`

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# In the examples below, ENC values are calculated for all sequences;
# SCUO values can be caluclated in the same way.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

# calculate ENC
enc <- ENC(cT)
head(enc)

# filtering
ENC(cT, filtering = "hard", len.threshold = 80) # ENC for 9 sequences
sum(getlen(cT) > 80) # 9 sequences are longer than 80 codons
enc1 <- ENC(cT, filtering = "none") # no filtering
enc2 <- ENC(cT, filtering = "soft") # warning
all.equal(enc1, enc2) # TRUE

# options for genetic code
enc <- ENC(cT, stop.rm = TRUE) # don't use stop codons in calculation
```

```
enc <- ENC(cT, alt.init = FALSE) # don't use alternative start codons
enc <- ENC(cT, id_or_name2 = "2") # use different genetic code, for help
                                  # see `?Biostrings::GENETIC_CODE`
```

---

codonUsage-expressivity

Calculate CU expressivity measures.

Description

Calculate values of the CU expressivity measure for every sequence in the given codonTable object. The following methods are implemented: MELP, CU expressivity measure based on Measure Independent of Length and Composition Supek & Vlahovicek (2005), E, gene expression measure (E) Karlin and Mrazek (2000), CAI, Codon Adaptation Index (CAI) Sharp and Li (1987), Fop, frequency of optimal codons (Fop) Ikemura (1981), GCB, gene codon bias (GCB) Merkl (2003).

Usage

```
MELP(cTobject, subsets = list(), ribosomal = FALSE,
  id_or_name2 = "1", alt.init = TRUE, stop.rm = FALSE,
  filtering = "none", len.threshold = 80)

## S4 method for signature 'codonTable'
MELP(cTobject, subsets = list(),
  ribosomal = FALSE, id_or_name2 = "1", alt.init = TRUE,
  stop.rm = FALSE, filtering = "none", len.threshold = 80)

E(cTobject, subsets = list(), ribosomal = FALSE, id_or_name2 = "1",
  alt.init = TRUE, stop.rm = FALSE, filtering = "none",
  len.threshold = 80)

## S4 method for signature 'codonTable'
E(cTobject, subsets = list(), ribosomal = FALSE,
  id_or_name2 = "1", alt.init = TRUE, stop.rm = FALSE,
  filtering = "none", len.threshold = 80)

CAI(cTobject, subsets = list(), ribosomal = FALSE, id_or_name2 = "1",
  alt.init = TRUE, stop.rm = FALSE, filtering = "none",
  len.threshold = 80)

## S4 method for signature 'codonTable'
CAI(cTobject, subsets = list(),
  ribosomal = FALSE, id_or_name2 = "1", alt.init = TRUE,
  stop.rm = FALSE, filtering = "none", len.threshold = 80)

Fop(cTobject, subsets = list(), ribosomal = FALSE, id_or_name2 = "1",
  alt.init = TRUE, stop.rm = FALSE, filtering = "none",
  len.threshold = 80)

## S4 method for signature 'codonTable'
```

```
Fop(cTobject, subsets = list(),
  ribosomal = FALSE, id_or_name2 = "1", alt.init = TRUE,
  stop.rm = FALSE, filtering = "none", len.threshold = 80)

GCB(cTobject, seed = logical(), ribosomal = FALSE, perc = 0.05,
  id_or_name2 = "1", alt.init = TRUE, stop.rm = FALSE,
  filtering = "none", len.threshold = 80)

## S4 method for signature 'codonTable'
GCB(cTobject, seed = logical(),
  ribosomal = FALSE, perc = 0.05, id_or_name2 = "1",
  alt.init = TRUE, stop.rm = FALSE, filtering = "none",
  len.threshold = 80)
```

Arguments

`cTobject`	A `codonTable` object.
`subsets`	A (named) list of logical vectors, the length of each equal to `getlen(cTobject)`, i.e. the number of sequences in the set, or character vectors (of any length) containing KEGG/eggNOG annotations, or codonTable objects (of any length). Not used for `ENC`, `SCUO` and `GCB` calculations.
`ribosomal`	Logical, if `TRUE`, CU statistic is also calculated against the average CU of the ribosomal genes in the sequence set. Not used for `ENC` and `SCUO` calculations. For GCB calculations, if `TRUE`, ribosomal genes are used as a seed, and if `FALSE` (default), `seed` has to be specified.
`id_or_name2`	A single string that uniquely identifies the genetic code to extract. Should be one of the values in the `id` or `name2` columns of `GENETIC_CODE_TABLE`.
`alt.init`	logical, whether to use alternative initiation codons. Default is `TRUE`.
`stop.rm`	Logical, whether to remove stop codons. Default is `FALSE`.
`filtering`	Character vector, one of `c("none", "soft", "hard")`. Specifies whether sequences shorther than some threshold value of length (in codons), `len.threshold`, should be excluded from calculations. If `"none"` (default), length of sequences is not checked, if `"soft"`, a warrning is printed if there are shorter sequences, and if `"hard"`, these sequences are excluded from calculation.
`len.threshold`	Optional numeric, specifying sequence length, in codons, used for filtering.
`seed`	A logical vector, of the length equal to `getlen(cTobject)`, or a character vector (of any length) containing KEGG/eggNOG annotations, or a codonTable object (of any length). Used only in GCB calculation. Indicates a set of genes, or their CU, to be used as a target in the first iteration of the algorithm.
`perc`	percent of top ranking genes to be used as a target set for the next iteration of the algorithm that calculates GCB. Default is 0.05.

Value

A matrix (for GCB a numeric vector) with CU expressivity values for every specified subset (`subsets`, `self`, `ribosomal`) in columns.

Examples

```
# load example DNA sequences
exampledir <- system.file("extdata", package = "coRdon")
```

```
cT <- codonTable(readSet(exampledir))

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# In the examples below, MELP values are calculated for all sequences;
# any other CU expressivity measure can be caluclated in the same way,
# the only exception being GCB which takes `seed` instead of `subset`
# parameter. (The exemples for GCB calculation are further below).
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

# calculate MELP with respect to the CU
# of ribosomal genes among the example DNA sequences
melp <- MELP(cT, ribosomal = TRUE)
head(melp)

# calculate MELP distance with respect to the average CU
# of the first 20 example DNA sequences
# (i.e. the first half of the example DNA set)
melp <- MELP(cT, subsets = list(half = c(rep(TRUE, 20), rep(FALSE, 20))))

# alternatively, you can specify codonTable as a subset
halfcT <- codonTable(codonCounts(cT)[1:20,])
melp2 <- MELP(cT, subsets = list(half = halfcT))
all.equal(melp, melp2) # TRUE

# filtering
MELP(cT, ribosomal = TRUE,
     filtering = "hard", len.threshold = 80) # MELP for 9 sequences
                                             # (note that, accidentally,
                                             # all are ribosomal)
sum(getlen(cT) > 80) # 9 sequences are longer than 80 codons
melp1 <- MELP(cT, ribosomal = TRUE, filtering = "none") # no filtering
melp2 <- MELP(cT, ribosomal = TRUE, filtering = "soft") # warning
all.equal(melp1, melp2) # TRUE

# options for genetic code
melp <- MELP(cT, ribosomal = TRUE,
             stop.rm = TRUE) # don't use stop codons in calculation
melp <- MELP(cT, ribosomal = TRUE,
             alt.init = FALSE) # don't use alternative start codons
melp <- MELP(cT, ribosomal = TRUE,
             id_or_name2 = "2") # use different genetic code, for help
                                # see `?Biostrings::GENETIC_CODE`

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# GCB calculationd
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

# calculate GCB with CU of ribosomal genes among the example DNA sequences
# used as a target (seed) in the first iteration of the algorithm
gcb <- GCB(cT, ribosomal = TRUE)
head(gcb)

# calculate GCB distance with the first 20 example DNA sequences
# (i.e. the first half of the example DNA set) as a seed
gcb <- GCB(cT, seed = c(rep(TRUE, 20), rep(FALSE, 20)))

# alternatively, you can specify codonTable as a seed
```

```
halfcT <- codonTable(codonCounts(cT)[1:20,])
gcb2 <- GCB(cT, seed = halfcT)
all.equal(gcb, gcb2) # TRUE

# options for genetic code
gcb <- GCB(cT, ribosomal = TRUE,
           stop.rm = TRUE) # don't use stop codons in calculation
gcb <- GCB(cT, ribosomal = TRUE,
           alt.init = FALSE) # don't use alternative start codons
gcb <- GCB(cT, ribosomal = TRUE,
           id_or_name2 = "2") # use different genetic code, for help
                              # see `?Biostrings::GENETIC_CODE`
```

---

coRdon *coRdon: codon usage analysis in R*

Description

R package for analysis of codone usage in unannotated or KEGG/COG annotated DNA sequences. Calculates various measures of CU bias and CU-based predictors of gene expression, and performs gene set enrichment analysis for annotated sequences. Implements several methods for visualization of CU and enrichment analysis results.

---

crossTab-class *An S4 class* crossTab

Description

Contingency table of sequences' annotations and the corresponding numeric values.

Usage

```
crossTab(sequences, variable, threshold = 1L, percentiles = NULL)

## S4 method for signature 'character,numeric'
crossTab(sequences, variable,
  threshold = 1L, percentiles = NULL)

getSeqAnnot(x)

## S4 method for signature 'crossTab'
getSeqAnnot(x)

getVariable(x)

## S4 method for signature 'crossTab'
getVariable(x)

contable(x)

## S4 method for signature 'crossTab'
contable(x)
```

Arguments

`sequences`	Character vector of sequences' annotations (KO, COG).
`variable`	Numeric vector of the coresponding CU values.
`threshold`	A threshold value (or a vector of values) of the variable. Sequences with value of the given variable greater than threshold are taken as a subset. Default is 1. If no threshold should be set, specify `threshold = NULL`
`percentiles`	A single value or a vector of values between 0 and 1. Sequences with value of the given variable in the top percentiles are taken as a subset. If no percentiles should be specified, the argument takes the value `NULL`.
`x`	A `crossTab` object.

Value

Returns a `crossTab` object with category values in rows, and with separate columns for counts in background (all) and subsets, i.e. for diferrent thresholds/percentiles provided.

Methods (by generic)

- `crossTab`: Create a contingency table for the set of annotated sequences and the corresponding codon usage (CU) values.
- `getSeqAnnot`: Get sequence annotations from `crossTab` object.
- `getVariable`: Get values of the variable used to create contingency table in `crossTab` object.
- `contable`: Get contingency table from `crossTab` object.

Slots

`sequences` Character vector of sequences annotations.

`variable` Numeric vector of the coresponding CU values.

`table` Contingecy table.

Examples

```
set.seed(5491)
s <- sample(LETTERS[1:3], 10, replace = TRUE)
v <- sample(1:5, 10, replace = TRUE)
crossTab(s, v)
crossTab(s, v, threshold = c(3,5))
crossTab(s, v, threshold = NULL, percentiles = c(0.5, 0.3))
ct <- crossTab(s, v)
contable(ct)
getSeqAnnot(ct)
getVariable(ct)
```

enrichBarplot *Barplot of enriched and depleted annotations.*

Description

Make a barplot of enriched annotations. Bars' heights represent values of the chosen enrichment statistic (c("enrich","M","A")), and the colours represent the p values (c("pvals", "padj")).

Usage

```
enrichBarplot(x, variable, pvalue = "pvals", siglev = numeric())

## S4 method for signature 'list'
enrichBarplot(x, variable, pvalue = "pvals",
  siglev = numeric())

## S4 method for signature 'AnnotatedDataFrame'
enrichBarplot(x, variable,
  pvalue = "pvals", siglev = numeric())
```

Arguments

x	AnnotatedDataFrame object, or a list of those.
variable	Character, indicating the statistic values to be used for plotting, must be one of c("enrich","M","A").
pvalue	Character, one of c("pvals", "padj").
siglev	Numeric, significance level to be used for plotting.

Value

A ggplot object.

Examples

```
require(ggplot2)

HD59_PATHWAYS
enrichBarplot(HD59_PATHWAYS, variable = "M",
              pvalue = "padj", siglev = 0.01) +
   labs(y = "pathway count\nlog ratios", x = "KEGG Pathway")

x <- list(disease = LD94_PATHWAYS, healthy = HD59_PATHWAYS)
enrichBarplot(x, variable = "enrich", pvalue = "padj", siglev = 0.01) +
    labs(y = "relative enrichment", x = "KEGG Pathway")
```

enrichMAplot *MA plot of enriched annotations.*

Description

Make an MA-like plot of enriched annotations, similar to the commonly used plots in differential expression analysis.

Usage

```
enrichMAplot(x, pvalue = "pvals", siglev = 0.05, size = 1,
  alpha = 1)

## S4 method for signature 'list'
enrichMAplot(x, pvalue = "pvals", siglev = 0.05,
  size = 1, alpha = 1)

## S4 method for signature 'AnnotatedDataFrame'
enrichMAplot(x, pvalue = "pvals",
  siglev = 0.05, size = 1, alpha = 1)
```

Arguments

x	AnnotatedDataFrame object, or a list of those.
pvalue	Character, one of c("pvals", "padj").
siglev	Numeric, significance level to be used for plotting.
size	Numeric, size of points in plot.
alpha	Numeric, between 0 and 1, indicating points' transparency.

Value

A ggplot object.

Examples

```
require(ggplot2)

HD59_KO
enrichMAplot(HD59_KO)
enrichMAplot(HD59_KO, pvalue = "padj")
enrichMAplot(HD59_KO, siglev = 0.01)
enrichMAplot(HD59_KO, pvalue = "padj", siglev = 0.01)

x <- list(disease = LD94_KO, healthy = HD59_KO)
enrichMAplot(x)
```

enrichMatrix *Extract chosen enrichment values to a matrix.*

Description

Extract enrichment values from multiple samples, i.e. AnnotatedDataFrame objects. Note that the samples should contain annotations of the same type (i.e. the same ontology). The data in matrix format can be easily used in different types of downstream analyses, such as GAGE, and visualised, e.g. using a heatmap.

Usage

```
enrichMatrix(x, variable, replace.na = TRUE)

## S4 method for signature 'list'
enrichMatrix(x, variable, replace.na = TRUE)
```

Arguments

x	A named list of AnnotatedDataFrame objects.
variable	Character, indicating the statistic values to extract from AnnotatedDataFrame objects in x, must be one of c("enrich","M","A").
replace.na	logical, whether to replace NA values in the output. If 'TRUE' (default), NAs will be replaced by 0. Alternatively, if numueric, NAs will be replaced by that given value.

Value

matrix with sequences' annotations as rows, and variable values for different samples as columns.

Examples

```
require(Biobase)

# create contingency table
s <- getKO(LD94)
v <- as.numeric(MELP(LD94, ribosomal = TRUE))
ct <- crossTab(s, v, percentiles = 0.2)

# enrichment analysis
enr <- enrichment(ct)
enr # for help, see `?Biobase::AnnotatedDataFrame`
head(pData(enr$top_0.2), 10)
head(pData(enr$gt_1), 10)
enrm <- enrichMatrix(enr, "M")
head(enrm)
```

enrichment *Enrichment analysis for codon usage (CU) data.*

Description

Performs enrichment analysis, given a contongency table of codon counts. p values are calculated by binomial test, adjustment for multiple testing can be performed by any of the `p.adjust.methods`.

Usage

```
enrichment(x, pvalueCutoff = numeric(), pAdjustMethod = "BH",
  padjCutoff = numeric())

## S4 method for signature 'crossTab'
enrichment(x, pvalueCutoff = numeric(),
  pAdjustMethod = "BH", padjCutoff = numeric())
```

Arguments

`x`	A `crossTab` object
`pvalueCutoff`	Numeric, discard categories with p value below this threshold. By default, no threshold is set (`numeric()`).
`pAdjustMethod`	Character, one of the `p.adjust.methods`.
`padjCutoff`	Numeric, discard categories with adjusted p value below this threshold. By default, no threshold is set (`numeric()`).

Value

An `AnnotatedDataFrame` object, or a list of those; data in each object has category values in rows, and the following columns:

- category, a character vector of annotation categories
- all, a numeric vector of integers, coresponding to sequence counts for each annotation category, in the background gene set (universe).
- a numeric vector(s) of integers, coresponding to sequence counts for each annotation category, in the set of genes for which enrichment is calculated, i.e. the predefined subset of (usually highly expressed) genes in the universe (named for the corresponding ‘crossTab‘ column).
- enrichment, calculated as the ratio: (scaled sample counts - scaled backg. counts) / scaled backg. counts * 100, where scaling means that sample counts are simply increased by 1, and background counts are multiplied by ratio of summed sample counts and summed backgroun counts, and also increased by 1
- M, log ratios of scaled counts
- A, mean average of scaled counts
- pvals, p values for exact binomial test
- padj, p values corrected by BH method.

Examples

```
require(Biobase)

# create contingency table
s <- getKO(HD59)
v <- as.numeric(MELP(HD59, ribosomal = TRUE))
ct <- crossTab(s, v)

# enrichment analysis
enr <- enrichment(ct)
enr # for help, see `?Biobase::AnnotatedDataFrame`
head(pData(enr))

enr <- enrichment(ct, pAdjustMethod = "holm")
head(pData(enr))

enr <- enrichment(ct, pvalueCutoff = 0.05)
head(pData(enr))

enr <- enrichment(ct, padjCutoff = 0.05)
head(pData(enr))
```

---

genCode-class *An S4 class* genCode

Description

Object of genCode class describes the variant of genetic code to be used in CU calculations.

Usage

```
genCode(id_or_name2 = "1", alt.init = TRUE, stop.rm = FALSE)

## S4 method for signature 'ANY'
genCode(id_or_name2 = "1", alt.init = TRUE,
  stop.rm = FALSE)
```

Arguments

id_or_name2	A single string that uniquely identifies the genetic code to extract. Should be one of the values in the id or name2 columns of GENETIC_CODE_TABLE.
alt.init	logical, whether to use alternative initiation codons. Default is TRUE.
stop.rm	logical, whether to remove stop codons. Default is FALSE.

Value

A genCode object.

Methods (by generic)

- genCode: Creates new instances of genCode class.

Slots

`ctab` A `data.table` with two colums: `codon` and `AA`, amino acid.

`codons` A character vector of codons.

`stops` A character vector of stop codons. Note that, if `stop.rm` is `TRUE`, this will be an empty vector.

`nostops` A character vector of no-stop codons. If `stop.rm` is `TRUE`, this will be equal to the `codons` slot.

`cl` A list, each element of which is a vector of integers indicating the positions of synonymous codons for that amino acid, when codons are ordered alphabetically.

`deg` A numeric vector of degeneracies for alphabetically ordered amino acids.

HD59 — *Codon usage in healthy human gut microbiome.*

Description

A `codonTable` object with codon counts for sequences of the human gut metagenome, from a healthy individual. Raw sequences are from Quin et al. 2014, processed, assembled and annotated (KEGG Orthology) as descrbed in Fabijanic and Vlahovicek 2016. Due to size limitations, a sample of 1000 sequences from the original data is used.

Usage

```
HD59
```

Format

A `codonTable` object.

Source

Quin et al. 2014; Fabijanic and Vlahovicek 2016

HD59_KO — *Codon usage based KO enrichment analysis results from the healthy human gut microbiome. For more information, see '?HD59'.*

Description

Codon usage based KO enrichment analysis results from the healthy human gut microbiome. For more information, see '?HD59'.

Usage

```
HD59_KO
```

Format

An `AnnotatedDataFrame` object. See '?enrichment' for description.

Source

Quin et al. 2014; Fabijanic and Vlahovicek 2016

HD59_PATHWAYS — *Codon usage based KEGG Pathway enrichment analysis results from a healthy human gut microbiome. For more information, see '?HD59'.*

Description

Codon usage based KEGG Pathway enrichment analysis results from a healthy human gut microbiome. For more information, see '?HD59'.

Usage

```
HD59_PATHWAYS
```

Format

An `AnnotatedDataFrame` object. See '?enrichment' for description.

Source

Quin et al. 2014; Fabijanic and Vlahovicek 2016

---

intraBplot — *Intra-samples Karlin B plot*

Description

Plot CU frequency distances between two samples (given by `x` and `y`).

Usage

```
intraBplot(x, y, names = c("x", "y"), variable, ribosomal = FALSE,
  size = 1, alpha = 0.5)

## S4 method for signature 'codonTable,codonTable'
intraBplot(x, y, names = c("x", "y"),
  variable, ribosomal = FALSE, size = 1, alpha = 0.5)
```

Arguments

`x, y`	Objects of `codonTable` class.
`names`	Character vector of length 2, giving names for samples.
`variable`	A character, name of the function that will be used to calculate CU statistic values for plotting. Must be one of the following: `c("MILC", "B", "MCB", "ENCprime")`.
`ribosomal`	Logical, whether to indicate ribosomal genes in the plot. Default is `FALSE`.
`size`	Numeric, indicating points' size
`alpha`	Numeric, between 0 and 1, indicating points' transparency (default is 0.1).

Value

A ggplot object.

Examples

```
require(ggplot2)
# calculate MILC distance to the average CU of the example DNA sequences,
# and to the average CU of ribosomal genes among the example DNA sequences
milc <- MILC(LD94, self = TRUE, ribosomal = TRUE)

intraBplot(x = HD59, y = LD94, names = c("HD59", "LD94"),
           variable = "MILC", size = 3)
```

LD94 — *Codon usage in human gut microbiome in liver cirrhosis.*

Description

A codonTable object with codon counts for sequences of the human gut metagenome, from an individual with liver cirrhosis. Raw sequences are from Quin et al. 2014, processed, assembled and annotated (KEGG Orthology) as descrbed in Fabijanic and Vlahovicek 2016. Due to size limitations, only a sample of 1000 sequences from the original data is used.

Usage

```
LD94
```

Format

A codonTable object.

Source

Quin et al. 2014; Fabijanic and Vlahovicek 2016

LD94_KO — *Codon usage based KO enrichment analysis results from an gut microbiome of an individual with liver cirrhosis. For more information, see '?LD94'.*

Description

Codon usage based KO enrichment analysis results from an gut microbiome of an individual with liver cirrhosis. For more information, see '?LD94'.

Usage

```
LD94_KO
```

Format

An AnnotatedDataFrame object. See ‘?enrichment‘ for description.

Source

Quin et al. 2014; Fabijanic and Vlahovicek 2016

LD94_PATHWAYS — *Codon usage based KEGG Pathway enrichment analysis results from an gut microbiome of an individual with liver cirrhosis. For more information, see ‘?LD94‘.*

Description

Codon usage based KEGG Pathway enrichment analysis results from an gut microbiome of an individual with liver cirrhosis. For more information, see ‘?LD94‘.

Usage

```
LD94_PATHWAYS
```

Format

An AnnotatedDataFrame object. See ‘?enrichment‘ for description.

Source

Quin et al. 2014; Fabijanic and Vlahovicek 2016

---

length-codonTable — *Length of* codonTable *object.*

Description

Length of codonTable object is the number of sequences for which there are codon counts contained in the object.

Usage

```
## S4 method for signature 'codonTable'
length(x)
```

Arguments

x	A codonTable object.

Value

Numeric, the length of x.

length-crossTab — *Length of* crossTab *object.*

Description

The length of crossTab is number of sequences for which the contingency table is contained in the object.

Usage

```
## S4 method for signature 'crossTab'
length(x)
```

Arguments

x	A crossTab object.

Value

Numeric, the length of x.

readSet — *Read set of sequences.*

Description

Reads a set of fasta files stored in folder, or a single fasta file.

Usage

```
readSet(folder = character(), file = character(), KOs = c(),
  zipped = FALSE, prepend.filenames = FALSE)
```

Arguments

folder	Path to directory containing .fasta files.
file	Path to a single .fasta file, or zipped file (if latter, specify ZIPPED = TRUE).
KOs	An optional character vector of sequence annotations (e.g. KO) contained in the names of fasta files to be selectively read.
zipped	Logical, whether folder or file is zipped. Default is FALSE.
prepend.filenames	Logical, whether to prepend filename(s) to names in DNAStringSet object. Default is FALSE.

Value

Returns a DNAStringSet object.

Examples

```
exampledir <- system.file("extdata", package = "coRdon")
files <- list.files(exampledir)
readSet(folder = exampledir)
readSet(folder = exampledir, KOs = "K02931")
pathtofile <- paste(exampledir, files[1], sep = "/")
readSet(file = pathtofile)
```

---

reduceCrossTab *Reduce* crossTab.

Description

Reduce the input contingency table by associating sequences with KEGG Pathway, KEGG Module or COG functional category identifiers.

Usage

```
reduceCrossTab(x, target)

## S4 method for signature 'crossTab,character'
reduceCrossTab(x, target)
```

Arguments

x	A crossTab object to be reduced.
target	Character vector indicating which onthology to use, either "pathway" or "module", or "cogfunction".

Value

Returns input crossTab object, with updated contingency table, displaying new category values in rows, and updated counts in columns.

Examples

```
# create contingency table
s <- getKO(HD59)
v <- as.numeric(MELP(HD59, ribosomal = TRUE))
ct <- crossTab(s, v)
ct

# reduce contingency table
reduceCrossTab(ct, "pathway")
reduceCrossTab(ct, "module")
```

RPKOs — *KEGG Orthology (KO) annotations for ribosomal genes.*

Description

KEGG Orthology (KO) annotations for ribosomal genes.

Usage

```
RPKOs
```

Format

A character vector.

show-codonTable — *Display the object of* codonTable *class.*

Description

Display the object of codonTable class.

Usage

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

Arguments

object	A codonTable object.

Value

show returns an invisible NULL.

show-crossTab — *Display the object of* crossTab *class.*

Description

Display the object of crossTab class.

Usage

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

Arguments

object	A crossTab object.

Value

show returns an invisible NULL.

[,codonTable-method *Subset* codonTable *object.*

Description

Subset codonTable object.

Usage

```
## S4 method for signature 'codonTable'
x[i]

## S4 method for signature 'codonTable'
x[[i]]

## S3 method for class 'codonTable'
subset(x, subset, ...)
```

Arguments

x	A codonTable object to be subset.
i	indices specifying elements to extract or replace. Indices are numeric or character vectors or empty (missing) or NULL. Numeric values are coerced to integer as by as.integer (and hence truncated towards zero). Character vectors will be matched to the names of the object (or for matrices/arrays, the dimnames): see 'Character indices' below for further details. For [-indexing only: i, j, ... can be logical vectors, indicating elements/slices to select. Such vectors are recycled if necessary to match the corresponding extent. i, j, ... can also be negative integers, indicating elements/slices to leave out of the selection. When indexing arrays by [a single argument i can be a matrix with as many columns as there are dimensions of x; the result is then a vector with elements corresponding to the sets of indices in each row of i. An index value of NULL is treated as if it were integer(0).
subset	A logical or character vector indicating which elements of x to keep. If logical, subset should be of length length(x). If character, subset should contain at least some of the elements of either getKO(x) or getCOG(x).
...	further arguments to be passed to or from other methods.

Value

subsets of codonTable object, keeping in each slot only those elements that meet the criteria in subset, if specified.

Examples

```
# create codonTable
mat <- matrix(sample(1:10, 610, replace = TRUE), nrow = 10)
cT <- codonTable(mat) # produces informative warning
cT
cT[1]
cT[[1]]
subset(cT, c(rep(c(TRUE,FALSE), 5))) # subset odd sequences

cT <- setKO(cT, rep(c("K00001", "K00002"), 5))
subset(cT, "K00001")

cT <- setCOG(cT, rep(c("COG0001", "COG0002"), 5))
subset(cT, "COG0001")
```

Index