Package 'cubar'

May 8, 2026

Title Codon Usage Bias Analysis

Version 1.2.0

Description A suite of functions for rapid and flexible analysis of codon usage bias. It provides in-depth analysis at the codon level, including relative synonymous codon usage (RSCU), tRNA weight calculations, machine learning predictions for optimal or preferred codons, and visualization of codon-anticodon pairing. Additionally, it can calculate various gene-specific codon indices such as codon adaptation index (CAI), effective number of codons (ENC), fraction of optimal codons (Fop), tRNA adaptation index (tAI), mean codon stabilization coefficients (CSCg), and GC contents (GC/GC3s/GC4d). It also supports both standard and non-standard genetic code tables found in NCBI, as well as custom genetic code tables.

License MIT + file LICENSE

URL https://github.com/mt1022/cubar, https://mt1022.github.io/cubar/

BugReports https://github.com/mt1022/cubar/issues

Encoding UTF-8

LazyData true

LazyDataCompression bzip2

Imports Biostrings (>= 2.60.0), IRanges (>= 2.34.0), data.table (>= 1.14.0), ggplot2 (>= 3.3.5), rlang (>= 0.4.11)

Depends R (>= 4.1.0)

Suggests knitr, rmarkdown, testthat (>= 3.0.0), reticulate

VignetteBuilder knitr

RoxygenNote 7.3.2

Config/testthat/edition 3

NeedsCompilation no

Author Hong Zhang [aut, cre] (ORCID: <https://orcid.org/0000-0002-4064-9432>), Mengyue Liu [aut], Bu Zi [aut]

Maintainer Hong Zhang <mt1022.dev@gmail.com>

Repository CRAN

Date/Publication 2025-08-19 07:30:02 UTC

Contents

aa2codon *amino acids to codons*

Description

A data.frame of mapping from amino acids to codons

Usage

```
aa2codon
```

Format

a data.frame with two columns: amino_acid, and codon.

amino_acid amino acid corresponding to the codon

codon codon identity

Source

It is actually the standard genetic code.

Examples

```
aa2codon
```

ca_pairs *Generate codon-anticodon pairing relationship*

Description

`ca_pairs` show possible codon-anticodons pairings

Usage

```
ca_pairs(codon_table = get_codon_table(), domain = "Eukarya", plot = FALSE)
```

Arguments

`codon_table`	a table of genetic code derived from `get_codon_table` or `create_codon_table`.
`domain`	The taxonomic domain of interest. "Eukarya" (default), "Bacteria" or "Archaea".
`plot`	FALSE (default) or TRUE. Whether to keep the columns required for plotting.

Value

a data.table of codon-anticodon pairing information. The columns represent the pairing type, codon, corresponding anticodon, and the encoded amino acid when the argument "plot" is FALSE.

Examples

```
# get possible codon and anticodon pairings for the vertebrate mitochondrial genetic code
ctab <- get_codon_table(gcid = '2')
pairing <- ca_pairs(ctab)
head(pairing)
```

---

check_cds *Quality control and preprocessing of coding sequences*

Description

check_cds performs comprehensive quality control on coding sequences (CDS) by filtering sequences based on various criteria and optionally removing start or stop codons. This function ensures that sequences meet the requirements for downstream codon usage analysis.

Usage

```
check_cds(
  seqs,
  codon_table = get_codon_table(),
  min_len = 6,
  check_len = TRUE,
  check_start = TRUE,
  check_stop = TRUE,
  check_istop = TRUE,
  rm_start = TRUE,
  rm_stop = TRUE,
  start_codons = c("ATG")
)
```

Arguments

seqs	Input CDS sequences as a DNAStringSet or compatible object.
codon_table	Codon table matching the genetic code of the input sequences. Generated using get_codon_table() or create_codon_table().
min_len	Minimum CDS length in nucleotides (default: 6).
check_len	Logical. Check whether CDS length is divisible by 3 (default: TRUE).
check_start	Logical. Check whether CDSs begin with valid start codons (default: TRUE).
check_stop	Logical. Check whether CDSs end with valid stop codons (default: TRUE).

check_istop	Logical. Check for internal stop codons (default: TRUE).
rm_start	Logical. Remove start codons from the sequences (default: TRUE).
rm_stop	Logical. Remove stop codons from the sequences (default: TRUE).
start_codons	Character vector specifying valid start codons (default: "ATG").

Value

A DNAStringSet containing filtered and optionally trimmed CDS sequences that pass all quality control checks.

Examples

```
# Perform CDS sequence quality control for a sample of yeast genes
s <- head(yeast_cds, 10)
print(s)
check_cds(s)
```

---

codon_diff *Differential codon usage analysis*

Description

codon_diff takes two set of coding sequences and perform differential codon usage analysis.

Usage

```
codon_diff(seqs1, seqs2, codon_table = get_codon_table())
```

Arguments

seqs1	DNAStringSet, or an object that can be coerced to a DNAStringSet
seqs2	DNAStringSet, or an object that can be coerced to a DNAStringSet
codon_table	a table of genetic code derived from get_codon_table or create_codon_table.

Value

a data.table of the differential codon usage analysis. Global tests examine wthether a codon is used differently relative to all the other codons. Family tests examine whether a codon is used differently relative to other codons that encode the same amino acid. Subfamily tests examine whether a codon is used differently relative to other synonymous codons that share the same first two nucleotides. Odds ratio > 1 suggests a codon is used at higher frequency in seqs1 than in seqs2.

Examples

```
yeast_exp_sorted <- yeast_exp[order(yeast_exp$fpkm),]
seqs1 <- yeast_cds[names(yeast_cds) %in% head(yeast_exp_sorted$gene_id, 1000)]
seqs2 <- yeast_cds[names(yeast_cds) %in% tail(yeast_exp_sorted$gene_id, 1000)]
cudiff <- codon_diff(seqs1, seqs2)
```

`codon_optimize` *Optimize codon usage in coding sequences*

Description

`codon_optimize` redesigns a coding sequence by replacing each codon with its optimal synonymous alternative, while maintaining the same amino acid sequence. This function supports multiple optimization methods and is useful for improving protein expression in heterologous systems.

Usage

```
codon_optimize(
  seq,
  optimal_codons = optimal_codons,
  cf = NULL,
  codon_table = get_codon_table(),
  level = "subfam",
  method = "naive",
  num_sequences = 1,
  organism = NULL,
  envname = "cubar_env",
  attention_type = "original_full",
  deterministic = TRUE,
  temperature = 0.2,
  top_p = 0.95,
  match_protein = FALSE,
  spliceai = FALSE
)
```

Arguments

`seq`	A coding sequence as a DNAString object or any object that can be coerced to DNAString. The sequence should not include stop codons.
`optimal_codons`	A table of optimal codons as generated by `est_optimal_codons()`, containing optimality information for each codon.
`cf`	Matrix of codon frequencies from `count_codons()`. Required for "IDT" method to determine codon frequency distributions.
`codon_table`	A codon table defining the genetic code, derived from `get_codon_table()` or `create_codon_table()`.

`level`	Character string specifying optimization level: "subfam" (default, within codon subfamilies) or "amino_acid" (within amino acid groups). Required for "naive" and "IDT" methods.
`method`	Character string specifying the optimization algorithm: • `"naive"` (default): Simple replacement with optimal codons • `"IDT"`: Method from Integrated DNA Technologies tool • `"CodonTransformer"`: Neural network-based optimization
`num_sequences`	Integer. Number of different optimized sequences to generate (default: 1). For "CodonTransformer" with deterministic=FALSE, each sequence is independently sampled.
`organism`	Organism identifier (integer ID or string name) for "CodonTransformer" method. Must be from ORGANISM2ID in CodonUtils (e.g., "Escherichia coli general").
`envname`	Environment name for "CodonTransformer" method. Should match your conda environment name (default: "cubar_env").
`attention_type`	Attention mechanism type for "CodonTransformer": • `"original_full"` (default): Standard attention • `"block_sparse"`: Memory-efficient sparse attention
`deterministic`	Logical. For "CodonTransformer" method: • `TRUE` (default): Deterministic decoding (most likely tokens) • `FALSE`: Probabilistic sampling based on temperature
`temperature`	Numeric. Controls randomness in non-deterministic mode for "CodonTransformer". Lower values (0.2, default) are conservative; higher values (0.8) increase diversity. Must be positive.
`top_p`	Numeric. Nucleus sampling threshold (0-1) for "CodonTransformer". Only tokens with cumulative probability up to this value are considered. Default: 0.95.
`match_protein`	Logical. For "CodonTransformer", constrains predictions to maintain exact amino acid sequence. Recommended for unusual proteins or high temperature settings (default: FALSE).
`spliceai`	Logical. Whether to predict splice sites using SpliceAI (default: FALSE). Requires appropriate environment setup.

Value

The return type depends on parameters:

- Single DNAString: When num_sequences=1 and spliceai=FALSE
- DNAStringSet: When num_sequences>1 and spliceai=FALSE
- data.table: When spliceai=TRUE (includes sequences and splice predictions)

References

Fallahpour A, Gureghian V, Filion GJ, Lindner AB, Pandi A. CodonTransformer: a multispecies codon optimizer using context-aware neural networks. Nat Commun. 2025 Apr 3;16(1):3205.

Jaganathan K, Panagiotopoulou S K, McRae J F, et al. Predicting splicing from primary sequence with deep learning. Cell, 2019, 176(3): 535-548.e24.

Examples

```
cf_all <- count_codons(yeast_cds)
optimal_codons <- est_optimal_codons(cf_all)
seq <- 'ATGCTACGA'
# method "naive":
codon_optimize(seq, optimal_codons)
# method "IDT":
codon_optimize(seq, cf = cf_all, method = "IDT")
codon_optimize(seq, cf = cf_all, method = "IDT", num_sequences = 10)

# # The following examples requires pre-installation of python package SpliceAI or Codon
#  # Transformer. see the codon optimization vignette for further details.
# seq_opt <- codon_optimize(seq, method = "CodonTransformer",
#     organism = "Saccharomyces cerevisiae")
# seqs_opt <- codon_optimize(seq, method = "CodonTransformer",
#     organism = "Saccharomyces cerevisiae", num_sequences = 10,
#     deterministic =FALSE, temperature = 0.4)
# seqs_opt <- codon_optimize(seq, cf = cf_all, method = "IDT",
#     num_sequences = 10, spliceai = TRUE)
# seq_opt <- codon_optimize(seq, method = "CodonTransformer",
#     organism = "Saccharomyces cerevisiae", spliceai = TRUE)
```

---

count_codons *Count codon frequencies in coding sequences*

Description

count_codons tabulates the frequency of all 64 possible codons across input coding sequences. This function provides the foundation for most codon usage bias analyses in the cubar package.

Usage

```
count_codons(seqs, ...)
```

Arguments

seqs	Coding sequences as a DNAStringSet object, or compatible input that can be coerced to DNAStringSet.
...	Additional arguments passed to Biostrings::trinucleotideFrequency.

Value

A matrix where rows represent individual CDS sequences and columns represent the 64 possible codons. Each cell contains the frequency count of the corresponding codon in the respective sequence.

Examples

```
# Count codon frequencies across all yeast CDS sequences
cf_all <- count_codons(yeast_cds)
dim(cf_all)
cf_all[1:5, 1:5]

# Count codons for a single sequence
count_codons(yeast_cds[1])
```

`create_codon_table` *Create custom codon table from amino acid-codon mapping*

Description

`create_codon_table` generates a codon table from a user-defined data frame that maps codons to their corresponding amino acids. This function enables analysis of non-standard or artificial genetic codes not available in the NCBI genetic code collection.

Usage

```
create_codon_table(aa2codon)
```

Arguments

`aa2codon`	A data frame with two required columns: • `amino_acid`: Three-letter amino acid abbreviations (e.g., "Ala", "Arg") • `codon`: Corresponding three-nucleotide codon sequences

Value

A data.table with four columns:

- `aa_code`: Single-letter amino acid code
- `amino_acid`: Three-letter amino acid abbreviation
- `codon`: Three-nucleotide codon sequence
- `subfam`: Codon subfamily identifier (amino_acid_XY format)

Examples

```
# View the example amino acid to codon mapping
head(aa2codon)

# Create a custom codon table
custom_table <- create_codon_table(aa2codon = aa2codon)
head(custom_table)
```

est_aau *Estimate Amino Acid Usage Frequencies of CDSs.*

Description

Estimate Amino Acid Usage Frequencies of CDSs.

Usage

```
est_aau(cf, codon_table = get_codon_table())
```

Arguments

cf	matrix of codon frequencies as calculated by 'count_codons()'.
codon_table	codon_table a table of genetic code derived from `get_codon_table` or `create_codon_table`.

Value

a data.table with amino acid frequencies of CDSs. The columns include three-letter abbreviation of the amino acid, single-letter abbreviation, usage frequency of the amino acid in all sequences, and usage frequency proportion.

Examples

```
# estimate amino acid frequencies of yeast genes
cf_all <- count_codons(yeast_cds)
aau <- est_aau(cf_all)
print(aau)
```

est_csc *Estimate Codon Stabilization Coefficient*

Description

`get_csc` calculate codon occurrence to mRNA stability correlation coefficients (Default to Pearson's).

Usage

```
est_csc(
  seqs,
  half_life,
  codon_table = get_codon_table(),
  cor_method = "pearson"
)
```

Arguments

seqs	CDS sequences of all protein-coding genes. One for each gene.
half_life	data.frame of mRNA half life (gene_id & half_life are column names).
codon_table	a table of genetic code derived from get_codon_table or create_codon_table.
cor_method	method name passed to 'cor.test' used for calculating correlation coefficients.

Value

a data.table of codons and their CSCs. The columns include codon, codon stability coefficient, and correlation P-value.

References

Presnyak V, Alhusaini N, Chen YH, Martin S, Morris N, Kline N, Olson S, Weinberg D, Baker KE, Graveley BR, et al. 2015. Codon optimality is a major determinant of mRNA stability. Cell 160:1111-1124.

Examples

```
# estimate yeast mRNA CSC
est_csc(yeast_cds, yeast_half_life)
```

---

est_optimal_codons *Identify optimal codons using statistical modeling*

Description

est_optimal_codons identifies optimal codons within each codon family or amino acid group using binomial regression. Optimal codons are those whose usage correlates positively with high gene expression or negatively with codon usage bias (ENC), suggesting they are preferred for efficient translation.

Usage

```
est_optimal_codons(
  cf,
  codon_table = get_codon_table(),
  level = "subfam",
  gene_score = NULL,
  fdr = 0.001
)
```

Arguments

cf	A matrix of codon frequencies as calculated by `count_codons()`. Rows represent sequences and columns represent codons.
codon_table	A codon table defining the genetic code, derived from `get_codon_table()` or `create_codon_table()`.
level	Character string specifying the analysis level: "subfam" (default, analyzes codon subfamilies) or "amino_acid" (analyzes at amino acid level).
gene_score	A numeric vector of gene-level scores used to identify optimal codons. Length must equal the number of rows in `cf`. Common choices include: • Gene expression levels (RPKM, TPM, FPKM) - optionally log-transformed • Protein abundance measurements • Custom gene importance scores If not provided, the negative of ENC values will be used (lower ENC = higher bias).
fdr	Numeric value specifying the false discovery rate threshold for determining statistical significance of codon optimality (default depends on method).

Value

A data.table containing the input codon table with additional columns indicating codon optimality status, statistical significance, and effect sizes from the regression analysis. The columns include single-letter abbreviation of the amino acid, three-letter abbreviation, codon, codon subfamily, regression coefficient, regression P-value, Benjamini and Hochberg corrected Q-value, and indication of whether the codon is optimal.

References

Presnyak V, Alhusaini N, Chen YH, Martin S, Morris N, Kline N, Olson S, Weinberg D, Baker KE, Graveley BR, et al. 2015. Codon optimality is a major determinant of mRNA stability. Cell 160:1111-1124.

Examples

```
# perform binomial regression for optimal codon estimation
cf_all <- count_codons(yeast_cds)
codons_opt <- est_optimal_codons(cf_all)
codons_opt <- codons_opt[optimal == TRUE]
codons_opt
```

est_rscu *Estimate Relative Synonymous Codon Usage (RSCU)*

Description

est_rscu calculates the Relative Synonymous Codon Usage (RSCU) values for codons, which quantify the bias in synonymous codon usage. RSCU values indicate whether a codon is used more (>1) or less (<1) frequently than expected under uniform usage within its synonymous group.

Usage

```
est_rscu(
  cf,
  weight = 1,
  pseudo_cnt = 1,
  codon_table = get_codon_table(),
  level = "subfam",
  incl_stop = FALSE
)
```

Arguments

cf	A matrix of codon frequencies as calculated by count_codons(). Rows represent sequences and columns represent codons.
weight	A numeric vector of the same length as the number of sequences in cf, providing different weights for sequences when calculating codon frequencies. For example, gene expression levels. Default is 1 (equal weights).
pseudo_cnt	Numeric pseudo count added to avoid division by zero when few sequences are available for RSCU calculation (default: 1).
codon_table	A codon table defining the genetic code, derived from get_codon_table() or create_codon_table().
level	Character string specifying the analysis level: "subfam" (default, analyzes codon subfamilies) or "amino_acid" (analyzes at amino acid level).
incl_stop	Logical. Whether to include RSCU values for stop codons in the output (default: FALSE).

Value

A data.table containing the codon table with additional columns for RSCU analysis: usage frequency counts (cts), frequency proportions (prop), CAI weights (w_cai), and RSCU values (rscu). The table includes amino acid codes, full amino acid names, codons, and subfamily classifications.

References

Sharp PM, Tuohy TM, Mosurski KR. 1986. Codon usage in yeast: cluster analysis clearly differentiates highly and lowly expressed genes. Nucleic Acids Res 14:5125-5143.

Examples

```
# Calculate RSCU for all yeast genes
cf_all <- count_codons(yeast_cds)
rscu_all <- est_rscu(cf_all)
head(rscu_all)

# Calculate RSCU for highly expressed genes (top 500)
heg <- head(yeast_exp[order(-yeast_exp$fpkm), ], n = 500)
cf_heg <- count_codons(yeast_cds[heg$gene_id])
rscu_heg <- est_rscu(cf_heg)
head(rscu_heg)
```

---

est_trna_weight *Estimate tRNA weights for TAI calculation*

Description

`est_trna_weight` calculates tRNA weights for each codon based on tRNA availability and codon-anticodon pairing efficiency. These weights are used in tRNA Adaptation Index (TAI) calculations and reflect how well each codon is supported by the cellular tRNA pool.

Usage

```
est_trna_weight(
  trna_level,
  codon_table = get_codon_table(),
  domain = "Eukarya",
  s = NULL
)
```

Arguments

`trna_level`	A named numeric vector of tRNA expression levels or gene copy numbers. Names should be in the format "AminoAcid-Anticodon" (e.g., "Ala-GCA"). Each value represents the abundance of that tRNA species.
`codon_table`	A codon table defining the genetic code, derived from `get_codon_table()` or `create_codon_table()`.
`domain`	Character string specifying the taxonomic domain: "Eukarya" (default), "Bacteria", or "Archaea". This determines the codon-anticodon pairing rules and selection penalties. Specify either "domain" or "s".
`s`	A named list of selection penalties for non-Watson-Crick pairings. If provided, overrides the default domain-specific penalties. Specify either "domain" or "s".

Value

A data.table containing comprehensive tRNA weight information with columns:

- `aa_code`: Single-letter amino acid code
- `amino_acid`: Three-letter amino acid abbreviation
- `codon`: Codon sequence
- `subfam`: Codon subfamily identifier
- `anticodon`: Corresponding anticodon sequence
- `trna_id`: tRNA identifier (amino_acid-anticodon)
- `ac_level`: tRNA abundance level
- `W`: Absolute adaptiveness value
- `w`: Relative adaptiveness (normalized weight for TAI)

References

dos Reis M, Savva R, Wernisch L. 2004. Solving the riddle of codon usage preferences: a test for translational selection. Nucleic Acids Res 32:5036-5044.

Sabi R, Tuller T. 2014. Modelling the efficiency of codon-tRNA interactions based on codon usage bias. DNA Res 21:511-526.

Examples

```
# Calculate tRNA weights for yeast using gene copy numbers
yeast_trna_w <- est_trna_weight(yeast_trna_gcn)
head(yeast_trna_w)

# View the weight distribution
hist(yeast_trna_w$w, main = "Distribution of tRNA weights")
```

---

extract_trna_gcn *Extract tRNA gene copy numbers from nature tRNA sequences*

Description

`extract_trna_gcn` processes tRNA sequence data from GtRNADB to extract gene copy numbers for each tRNA type. This information is essential for calculating tRNA availability weights used in TAI analysis.

Usage

```
extract_trna_gcn(trna_seq)
```

Arguments

trna_seq	A named vector or DNAStringSet of tRNA sequences, typically from GtRNADB. Sequence names should follow the standard format containing amino acid and anticodon information (e.g., "tRNA-Ala-AGC-1-1").

Value

A named table of tRNA gene copy numbers. Names are in the format "AminoAcid-Anticodon" (e.g., "Ala-AGC") and values represent the count of genes encoding each tRNA type. Initiator tRNAs (iMet, fMet) and undetermined tRNAs (Und-NNN) are automatically excluded as they serve specialized functions in translation initiation.

Examples

```
# Extract tRNA gene copy numbers for yeast
trna_gcn <- extract_trna_gcn(yeast_trna)
head(trna_gcn)

# View the distribution of tRNA gene copies
hist(trna_gcn, main = "Distribution of tRNA gene copy numbers")
```

---

get_aau *Amino Acid Usage*

Description

Calculate Amino Acid Usage Frequencies of each CDS.

Usage

```
get_aau(cf, codon_table = get_codon_table())
```

Arguments

cf	matrix of codon frequencies as calculated by ‘count_codons()‘.
codon_table	a table of genetic code derived from `get_codon_table` or `create_codon_table`.

Value

a matrix of amino acid frequencies for each CDS. Each row corresponds to a sequence, and each column represents an amino acid.

Examples

```
# estimate amino acid frequencies of yeast CDSs
cf_all <- count_codons(yeast_cds)
aau_gene <- get_aau(cf_all)
head(aau_gene)
```

get_cai *Calculate Codon Adaptation Index (CAI)*

Description

get_cai calculates the Codon Adaptation Index (CAI) for each input coding sequence. CAI measures how similar the codon usage of a gene is to that of highly expressed genes, serving as an indicator of translational efficiency. Higher CAI values suggest better adaptation to the translational machinery.

Usage

```
get_cai(cf, rscu, level = "subfam")
```

Arguments

cf	A matrix of codon frequencies as calculated by count_codons(). Rows represent sequences and columns represent codons.
rscu	An RSCU table containing CAI weights for each codon. This table should be generated using est_rscu() based on highly expressed genes, or prepared manually with appropriate weight values.
level	Character string specifying the analysis level: "subfam" (default, analyzes codon subfamilies) or "amino_acid" (analyzes at amino acid level).

Value

A named numeric vector of CAI values ranging from 0 to 1. Names correspond to sequence identifiers from the input matrix. Values closer to 1 indicate higher similarity to highly expressed genes.

References

Sharp PM, Li WH. 1987. The codon Adaptation Index–a measure of directional synonymous codon usage bias, and its potential applications. Nucleic Acids Res 15:1281-1295.

Examples

```
# Calculate CAI for yeast genes based on RSCU of highly expressed genes
heg <- head(yeast_exp[order(-yeast_exp$fpkm), ], n = 500)
cf_all <- count_codons(yeast_cds)
cf_heg <- cf_all[heg$gene_id, ]
rscu_heg <- est_rscu(cf_heg)
cai <- get_cai(cf_all, rscu_heg)
head(cai)
hist(cai, main = "Distribution of CAI values")
```

get_codon_table *Retrieve codon table by NCBI genetic code ID*

Description

get_codon_table creates a standardized codon table based on genetic codes cataloged by NCBI. This function provides the mapping between codons and amino acids for different organisms and organelles, which is essential for accurate codon usage analysis.

Usage

```
get_codon_table(gcid = "1")
```

Arguments

gcid	A character string specifying the NCBI genetic code ID. Use show_codon_tables() to view all available genetic codes and their corresponding IDs. Default is "1" (standard genetic code).

Value

A data.table with four columns:

- aa_code: Single-letter amino acid code
- amino_acid: Three-letter amino acid abbreviation
- codon: Three-nucleotide codon sequence
- subfam: Codon subfamily identifier (amino_acid_XY format)

Examples

```
# Standard genetic code (used by most organisms)
standard_code <- get_codon_table()
head(standard_code)

# Vertebrate mitochondrial genetic code
mito_code <- get_codon_table(gcid = '2')
head(mito_code)
```

get_cscg *Mean Codon Stabilization Coefficients*

Description

get_cscg calculates Mean Codon Stabilization Coefficients of each CDS.

Usage

```
get_cscg(cf, csc)
```

Arguments

cf	matrix of codon frequencies as calculated by count_codons().
csc	table of Codon Stabilization Coefficients as calculated by est_csc().

Value

a named vector of cscg values. The names of the elements correspond to the sequence names.

References

Presnyak V, Alhusaini N, Chen YH, Martin S, Morris N, Kline N, Olson S, Weinberg D, Baker KE, Graveley BR, et al. 2015. Codon optimality is a major determinant of mRNA stability. Cell 160:1111-1124.

Examples

```
# estimate CSCg of yeast genes
yeast_csc <- est_csc(yeast_cds, yeast_half_life)
cf_all <- count_codons(yeast_cds)
cscg <- get_cscg(cf_all, csc = yeast_csc)
head(cscg)
hist(cscg)
```

get_dp *Deviation from Proportionality*

Description

get_dp calculates Deviation from Proportionality of each CDS.

Usage

```
get_dp(
  cf,
  host_weights,
  codon_table = get_codon_table(),
  level = "subfam",
  missing_action = "ignore"
)
```

Arguments

cf	matrix of codon frequencies as calculated by count_codons().
host_weights	a named vector of tRNA weights for each codon that reflects the relative availability of tRNAs in the host organism.
codon_table	a table of genetic code derived from get_codon_table or create_codon_table.
level	"subfam" (default) or "amino_acid". If "subfam", the deviation is calculated at the codon subfamily level. Otherwise, the deviation is calculated at the amino acid level.
missing_action	Actions to take when no codon of a group were found in a CDS. Options are "ignore" (default), or "zero" (set codon proportions to 0).

Value

a named vector of dp values. The names of the elements correspond to the sequence names.

References

Chen F, Wu P, Deng S, Zhang H, Hou Y, Hu Z, Zhang J, Chen X, Yang JR. 2020. Dissimilation of synonymous codon usage bias in virus-host coevolution due to translational selection. Nat Ecol Evol 4:589-600.

Examples

```
# estimate DP of yeast genes
cf_all <- count_codons(yeast_cds)
trna_weight <- est_trna_weight(yeast_trna_gcn)
trna_weight <- setNames(trna_weight$w, trna_weight$codon)
dp <- get_dp(cf_all, host_weights = trna_weight)
head(dp)
hist(dp)
```

get_enc *Calculate effective number of codons (ENC)*

Description

get_enc computes the effective number of codons (ENC) for each coding sequence, which quantifies the degree of codon usage bias. Lower ENC values indicate stronger bias (fewer codons are used), while higher values indicate more uniform codon usage.

Usage

```
get_enc(cf, codon_table = get_codon_table(), level = "subfam")
```

Arguments

cf	A matrix of codon frequencies as calculated by count_codons(). Rows represent sequences and columns represent codons.
codon_table	A codon table defining the genetic code, derived from get_codon_table() or create_codon_table().
level	Character string specifying the analysis level: "subfam" (default, analyzes codon subfamilies) or "amino_acid" (analyzes at amino acid level).

Value

A named numeric vector of ENC values. Names correspond to sequence identifiers from the input matrix. ENC values typically range from 20 (maximum bias) to 61 (uniform usage).

References

Wright F. 1990. The 'effective number of codons' used in a gene. Gene 87:23-29.

Sun X, Yang Q, Xia X. 2013. An improved implementation of effective number of codons (NC). Mol Biol Evol 30:191-196.

Examples

```
# Calculate ENC for yeast genes
cf_all <- count_codons(yeast_cds)
enc <- get_enc(cf_all)
head(enc)
hist(enc, main = "Distribution of ENC values")
```

get_fop *Calculate fraction of optimal codons (Fop)*

Description

get_fop calculates the fraction of optimal codons (Fop) for each coding sequence, which represents the proportion of codons that are considered optimal for translation efficiency. Higher Fop values suggest stronger selection for optimal codon usage.

Usage

```
get_fop(cf, op = NULL, codon_table = get_codon_table(), ...)
```

Arguments

cf	A matrix of codon frequencies as calculated by count_codons(). Rows represent sequences and columns represent codons.
op	A character vector specifying which codons are considered optimal. If not provided, optimal codons will be determined automatically using est_optimal_codons().
codon_table	A codon table defining the genetic code, derived from get_codon_table() or create_codon_table().
...	Additional arguments passed to est_optimal_codons() when optimal codons are determined automatically.

Value

A named numeric vector of Fop values (ranging from 0 to 1). Names correspond to sequence identifiers from the input matrix. Higher values indicate greater usage of optimal codons.

References

Ikemura T. 1981. Correlation between the abundance of Escherichia coli transfer RNAs and the occurrence of the respective codons in its protein genes: a proposal for a synonymous codon choice that is optimal for the E. coli translational system. J Mol Biol 151:389-409.

Examples

```
# Calculate Fop for yeast genes (optimal codons determined automatically)
cf_all <- count_codons(yeast_cds)
fop <- get_fop(cf_all)
head(fop)
hist(fop, main = "Distribution of Fop values")
```

get_gc *Calculate GC content of coding sequences*

Description

get_gc calculates the overall GC content (percentage of guanine and cytosine nucleotides) for each coding sequence.

Usage

```
get_gc(cf)
```

Arguments

cf	A matrix of codon frequencies as calculated by count_codons(). Rows represent sequences and columns represent codons.

Value

A named numeric vector of GC content values (ranging from 0 to 1). Names correspond to sequence identifiers from the input matrix.

Examples

```
# Calculate GC content for yeast genes
cf_all <- count_codons(yeast_cds)
gc <- get_gc(cf_all)
head(gc)
hist(gc, main = "Distribution of GC content")
```

get_gc3s *GC contents at synonymous 3rd codon positions*

Description

Calculate GC content at synonymous 3rd codon positions.

Usage

```
get_gc3s(cf, codon_table = get_codon_table(), level = "subfam")
```

Arguments

cf	matrix of codon frequencies as calculated by count_codons().
codon_table	a table of genetic code derived from get_codon_table or create_codon_table.
level	"subfam" (default) or "amino_acid". For which level to determine GC content at synonymous 3rd codon positions.

Value

a named vector of GC3s values. The names of the elements correspond to the sequence names.

References

Peden JF. 2000. Analysis of codon usage.

Examples

```
# estimate GC3s of yeast genes
cf_all <- count_codons(yeast_cds)
gc3s <- get_gc3s(cf_all)
head(gc3s)
hist(gc3s)
```

---

get_gc4d *GC contents at 4-fold degenerate sites*

Description

Calculate GC content at synonymous position of codons (using four-fold degenerate sites only).

Usage

```
get_gc4d(cf, codon_table = get_codon_table(), level = "subfam")
```

Arguments

cf	matrix of codon frequencies as calculated by count_codons().
codon_table	a table of genetic code derived from get_codon_table or create_codon_table.
level	"subfam" (default) or "amino_acid". For which level to determine GC contents at 4-fold degenerate sites.

Value

a named vector of GC4d values. The names of the elements correspond to the sequence names.

Examples

```
# estimate GC4d of yeast genes
cf_all <- count_codons(yeast_cds)
gc4d <- get_gc4d(cf_all)
head(gc4d)
hist(gc4d)
```

get_tai *Calculate tRNA Adaptation Index (TAI)*

Description

get_tai calculates the tRNA Adaptation Index (TAI) for each coding sequence, which measures how well codon usage matches tRNA availability in the cell. Higher TAI values indicate better adaptation to the tRNA pool, suggesting more efficient translation.

Usage

```
get_tai(cf, trna_w, w_format = "cubar")
```

Arguments

cf	A matrix of codon frequencies as calculated by count_codons(). Note: Start codons should be removed from sequences before analysis to avoid bias from universal start codon usage.
trna_w	A table of tRNA weights for each codon, generated using est_trna_weight(). These weights reflect relative tRNA availability.
w_format	Character string specifying the format of tRNA weights: "cubar" (default, weights from cubar package) or "tAI" (weights from the tAI package format).

Value

A named numeric vector of TAI values. Names correspond to sequence identifiers from the input matrix. Values range from 0 to 1, with higher values indicating better adaptation to tRNA availability. It should be noted that the tAI package does not include codons that correspond to methionine in its calculation. In contrast, cubar requires removing the start codon and takes into account codons encoding methionine within the sequence. Correspondingly, cubar excludes initiator tRNA (iMet in eukaryotes or fMet in prokaryotes), and focuses on the anticodons that correspond to regular methionine.

References

dos Reis M, Savva R, Wernisch L. 2004. Solving the riddle of codon usage preferences: a test for translational selection. Nucleic Acids Res 32:5036-5044.

Examples

```
# calculate TAI of yeast genes based on genomic tRNA copy numbers
w <- est_trna_weight(yeast_trna_gcn)
# note: check_istop is suppressed to facilitate package development.
#   We suggest enable this option for real sequence analyses.
yeast_cds_qc <- check_cds(yeast_cds, check_istop = FALSE)
cf <- count_codons(yeast_cds_qc)
tai <- get_tai(cf, w)
head(tai)
```

```
hist(tai)
```

---

human_mt *human mitochondrial CDS sequences*

Description

CDSs of 13 protein-coding genes in the human mitochondrial genome extracted from ENSEMBL Biomart

Usage

```
human_mt
```

Format

a DNAStringSet of 13 sequences

Source

<https://www.ensembl.org/index.html>

Examples

```
head(human_mt)
```

---

plot_ca_pairs *Plot codon-anticodon pairing relationship*

Description

`plot_ca_pairs` show possible codon-anticodons pairings

Usage

```
plot_ca_pairs(codon_table = get_codon_table(), pairs = pairs)
```

Arguments

`codon_table`	a table of genetic code derived from `get_codon_table` or `create_codon_table`.
`pairs`	a table of codon-anticodon pairing derived from `ca_pairs`

Value

a plot on possible codon-anticodons pairings

Examples

```
# plot possible codon and anticodon pairings for the vertebrate mitochondrial genetic code
ctab <- get_codon_table(gcid = '2')
pairs <- ca_pairs(ctab, plot = TRUE)
plot_ca_pairs(ctab, pairs)

# plot possible codon and anticodon pairings for the standard genetic code in bacteria
plot_ca_pairs(pairs = ca_pairs(domain = "Bacteria", plot = TRUE))
```

---

rev_comp *Generate reverse complement sequences*

Description

rev_comp generates the reverse complement of input DNA sequences. This is commonly used for analyzing complementary strands or anticodon sequences.

Usage

```
rev_comp(seqs)
```

Arguments

seqs	Input DNA sequences as a DNAStringSet object, or a named vector of sequences that can be coerced to DNAStringSet.

Value

A DNAStringSet object containing the reverse complemented sequences.

Examples

```
# Reverse complement of codons
rev_comp(Biostrings::DNAStringSet(c('TAA', 'TAG')))
```

seq_to_codons *Convert a coding sequence to a codon vector*

Description

seq_to_codons converts a coding sequence (CDS) into a vector of codons by splitting the sequence into non-overlapping triplets starting from the first position.

Usage

```
seq_to_codons(seq)
```

Arguments

seq	A coding sequence as a DNAString object, or any object that can be coerced to a DNAString (e.g., character string).

Value

A character vector where each element represents a codon (3-nucleotide sequence).

Examples

```
# Convert a CDS sequence to a sequence of codons
seq_to_codons('ATGTGGTAG')
seq_to_codons(yeast_cds[[1]])
```

show_codon_tables *Display available genetic code tables*

Description

show_codon_tables displays a formatted list of all genetic code tables available from NCBI, showing their ID numbers and descriptive names. This function helps users identify the appropriate genetic code ID to use with get_codon_table().

Usage

```
show_codon_tables()
```

Value

No return value (called for side effects). The function prints a formatted table of available genetic codes to the console, with each line showing the numeric ID and corresponding organism/organelle description.

Examples

```
# Display all available NCBI genetic code tables
show_codon_tables()
```

`slide` *Generate sliding window intervals*

Description

`slide` creates a data.table defining sliding window positions for analyzing sequences or data along a continuous range. This function provides the foundation for positional analyses of codon usage patterns within genes.

Usage

```
slide(from, to, step = 1, before = 0, after = 0)
```

Arguments

`from`	Integer specifying the start position of the analysis range.
`to`	Integer specifying the end position of the analysis range.
`step`	Integer specifying the step size between consecutive window centers (default: 1). Larger values create non-overlapping or less overlapping windows.
`before`	Integer specifying the number of positions to include before the window center (default: 0). Determines the left boundary of each window.
`after`	Integer specifying the number of positions to include after the window center (default: 0). Determines the right boundary of each window.

Value

A data.table with three columns:

- `start`: Start position of each window
- `center`: Center position of each window
- `end`: End position of each window

Examples

```
# Create sliding windows with step size 2 and window size 3
slide(1, 10, step = 2, before = 1, after = 1)
```

slide_apply *apply a cub index to a sliding window*

Description

slide_apply applies a function to a sliding window of codons.

Usage

```
slide_apply(seq, .f, step = 1, before = 0, after = 0, ...)
```

Arguments

seq	DNAString, the sequence
.f	function, the codon index calculation function to apply, for example, get_enc.
step	integer, the step size in number of codons
before	integer, the number of codons before the center of a window
after	integer, the number of codons after the center of a window
...	additional arguments to pass to the function .f

Value

data.table with start, center, end, and codon usage index columns

Examples

```
slide_apply(yeast_cds[[1]], get_enc, step = 1, before = 10, after = 10)
```

slide_codon *Generate sliding windows for codon-level analysis*

Description

slide_codon creates sliding window intervals specifically designed for codon-based analysis of DNA sequences. This function automatically handles codon boundaries and is useful for studying positional effects in codon usage within genes.

Usage

```
slide_codon(seq, step = 1, before = 0, after = 0)
```

Arguments

seq	A DNA sequence as a DNAString object, or any object that can be coerced to DNAString.
step	Integer specifying the step size between consecutive window centers in codons (default: 1). A step of 3 creates non-overlapping windows.
before	Integer specifying the number of codons to include before the window center (default: 0).
after	Integer specifying the number of codons to include after the window center (default: 0).

Value

A data.table with three columns containing nucleotide positions:

- start: Start nucleotide position of each window
- center: Center nucleotide position of each window
- end: End nucleotide position of each window

Examples

```
# Create sliding windows for codon analysis
x <- Biostrings::DNAString('ATCTACATAGCTACGTAGCTCGATGCTAGCATGCATCGTACGATCGTCGATCGTAG')
slide_codon(x, step = 3, before = 1, after = 1)
```

---

slide_plot *plot sliding window codon usage*

Description

slide_plot visualizes codon usage in sliding window.

Usage

```
slide_plot(windt, index_name = "Index")
```

Arguments

windt	data.table, the sliding window codon usage generated by slide_apply.
index_name	character, the name of the index to display.

Value

ggplot2 plot.

Examples

```
sw <- slide_apply(yeast_cds[[1]], get_enc, step = 1, before = 10, after = 10)
slide_plot(sw)
```

yeast_cds *yeast CDS sequences*

Description

CDSs of all protein-coding genes in Saccharomyces_cerevisiae

Usage

```
yeast_cds
```

Format

a DNAStringSet of 6600 sequences

Source

<https://ftp.ensembl.org/pub/release-107/fasta/saccharomyces_cerevisiae/cds/Saccharomyces_cerevisiae.R64-1-1.cds.all.fa.gz>

Examples

```
head(yeast_cds)
```

yeast_exp *yeast mRNA expression levels*

Description

Yeast mRNA FPKM determined from rRNA-depleted (RiboZero) total RNA-Seq libraries. RUN1_0_WT and RUN2_0_WT (0 min after RNA Pol II repression) were averaged and used here.

Usage

```
yeast_exp
```

Format

a data.frame with 6717 rows and three columns:

gene_id gene ID

gene_name gene name

fpkm mRNA expression level in Fragments per kilobase per million reads

Source

<https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE57385>

References

Presnyak V, Alhusaini N, Chen YH, Martin S, Morris N, Kline N, Olson S, Weinberg D, Baker KE, Graveley BR, et al. 2015. Codon optimality is a major determinant of mRNA stability. Cell 160:1111-1124.

Examples

```
head(yeast_exp)
```

`yeast_half_life` *Half life of yeast mRNAs*

Description

Half life of yeast mRNAs in Saccharomyces_cerevisiae calculated from rRNA-deleted total RNAs by Presnyak et al.

Usage

```
yeast_half_life
```

Format

a data.frame with 3888 rows and three columns:

gene_id gene id

gene_name gene name

half_life mRNA half life in minutes

Source

<https://doi.org/10.1016/j.cell.2015.02.029>

References

Presnyak V, Alhusaini N, Chen YH, Martin S, Morris N, Kline N, Olson S, Weinberg D, Baker KE, Graveley BR, et al. 2015. Codon optimality is a major determinant of mRNA stability. Cell 160:1111-1124.

Examples

```
head(yeast_half_life)
```

yeast_trna *yeast tRNA sequences*

Description

Yeast tRNA sequences obtained from gtRNAdb.

Usage

```
yeast_trna
```

Format

a RNAStringSet with a length of 275.

Source

<http://gtrnadb.ucsc.edu/genomes/eukaryota/Scere3/sacCer3-mature-tRNAs.fa>

References

Chan PP, Lowe TM. 2016. GtRNAdb 2.0: an expanded database of transfer RNA genes identified in complete and draft genomes. Nucleic Acids Res 44:D184-189.

Examples

```
yeast_trna
```

yeast_trna_gcn *yeast tRNA gene copy numbers (GCN)*

Description

Yeast tRNA gene copy numbers (GCN) by anticodon obtained from gtRNAdb.

Usage

```
yeast_trna_gcn
```

Format

a named vector with a length of 41. Value names are anticodons.

Source

<http://gtrnadb.ucsc.edu/genomes/eukaryota/Scere3/sacCer3-mature-tRNAs.fa>

References

Chan PP, Lowe TM. 2016. GtRNAdb 2.0: an expanded database of transfer RNA genes identified in complete and draft genomes. Nucleic Acids Res 44:D184-189.

Examples

```
yeast_trna_gcn
```

Index